LŐCS - VIGASSY

A FORTRAN PROGRAMOZÁSI NYELV

Műszaki Könyvkiadó

Josef Havel
Max - Stromeyer - Str. 9
7750 Konstanz

LŐCS GYULA – VIGASSY JÓZSEF

A FORTRAN programozási nyelv

LŐCS GYULA — VIGASSY JÓZSEF

A FORTRAN programozási nyelv

6., bővített kiadás

MŰSZAKI KÖNYVKIADÓ, BUDAPEST, 1985

Lektorálta:

GÖMBÖS ERVIN
matematika-fizika tanár

© Lőcs Gyula – Vigassy József, 1970

A *B)* Függeléket írták:

LOVAS ERIKA és GLADKIH IRINA

Lektorálta:

LŐCS GYULA
okl. matematikus

© Lovas Erika – Gladkih Irina, 1984

ETO: 681.3.06 FORTRAN
ISBN: 963 10 3845 9 (első kiadás)
ISBN: 963 10 6269 4

Felelős szerkesztő: Mikolás Zoltán okl. villamosmérnök

ELŐSZÓ

Az elektronikus számológépek elterjedése Magyarországon szükségszerűen felkeltette az érdeklődést a számológépek szakirodalma iránt, így speciálisan az ún. *magasabb szintű programozási nyelveket* ismertető leírások iránt is.

Műszaki és tudományos jellegű programozási feladatok megoldásával kapcsolatban eddig két univerzális programozási „világnyelv" terjedt el: az ALGOL 60 és a FORTRAN. Ezek közül magyar nyelven eddig csak az előbbiről jelent meg bevezető jellegű leírás. Azóta több oldalról is igény mutatkozott a FORTRAN nyelvről szóló hasonló jellegű munka iránt. Ezt az igényt szeretné kielégíteni ez a könyv.

A programozási nyelveket ismertető könyvekkel szemben háromféle követelményt szoktak támasztani. Ezek egyike a didaktikus felépítés: a nyelvet nem ismerő olvasó számára olyan módon előadni az egyes programozási „játékszabályokat", hogy azokat megértse, megtanulja és használatukat begyakorolja — tehát, hogy a könyvet *tankönyvként* használhassa.

Az előbbivel meglehetősen ellentétes szempontjai vannak az olyan olvasónak, aki a nyelvet már ismeri, és ezért a könyvet nem tankönyvként, hanem *kézikönyvként* kívánja használni. Ebből a szempontból a didaktikus felépítés kevésbé lényeges, annál inkább az anyag teljessége, a szabályok jól rendszerezett és áttekinthető tárgyalása. Erre a célra elvileg a programozási nyelvek hivatalos „jelentései" lennének a legmegfelelőbbek (pl. [1]), azonban egyrészt ezek nem mindig hozzáférhetők (különösen magyar nyelven nem), másrészt körmönfont, mindennemű magyarázatot nélkülöző, „hátsó gondolatokkal" agyonzsúfolt megfogalmazásaik miatt nem mindig kényelmesek.

Az előző két követelmény mellett az utóbbi időben egy további irányzat is jelentkezik: a *numerikus analízis és egyéb rokon tudományágak oktatása programozási úton.* Ebben az esetben tehát nem a programozási nyelv oktatásáról van szó, hanem a programozási nyelvnek mint eszköznek a felhasználásával a megfelelő szaktudomány oktatásáról. A lényeges különbség a hagyományos oktatási módszerrel szemben itt az, hogy az egyes eljárásokat nemcsak *formulák*, hanem egyúttal *programok* formájában is megfogalmazzák (lásd pl. [3], [4], [11]). Ehhez természetesen elkerülhetetlen valamelyik konkrét algoritmikus nyelv ismerete, azonban a hangsúly nem ezen van. Ennek az oktatási módszernek előnye, hogy — a numerikus ismereteken túlmenően — kifejleszti a hallgatóban az „algoritmus teremtés" készségét, amelyre egy holt képletnek élő programmá való „átvarázsolásához" van szükség. Tulajdonképpen ez az, amit *programozásnak* nevezünk, szemben az algoritmusok *kódolásával* — egy programozási nyelv szabályai szerint.

A számolóközpontok tapasztalata, amit a külföldi irodalom is egybehangzóan bizonyít, hogy valamely programozási nyelven *kódolni* néhány hét vagy hónap alatt meg lehet tanulni, de hogy valakiben kifejlődjék a *programozási* készség, ahhoz évek szükségesek.

Könyvünkben megkíséreltük összeegyeztetni az első szempontot a másodikkal, vagyis arra törekedtünk, hogy mind tankönyvként, mind kézikönyvként felhasználható legyen. Nem volt azonban szándékunk, hogy a könyvet „programozási szakácskönyvvé" tegyük, bár a példák egy része témáját a numerikus analízis területéről veszi. (Más részük kifejezetten adatfeldolgozási jellegű.) Az algoritmizálási készség fejlesztését — amennyire a terjedelem megengedte — szem előtt tartottuk; ebből a célból kísérjük végig pl. az utolsó fejezetben két bonyolultabb probléma útját a feladat kitűzésétől a megoldást realizáló gépi programig.

Bár fő célunk az általánosan elfogadott „standard" FORTRAN ismertetése volt, sokszor kívánatosnak látszott rámutatni ennek eltérésére több más FORTRAN változattól, egyes esetekben pedig az ALGOL 60-tól. Az ilyen megjegyzéseket apróbetűs szedéssel különítettük el. Ezek a részek első

olvasáskor elhagyhatók. Lényegesek viszont a programozási feladatok az egyes fejezetek végén, amelyek az anyag begyakorlásához nyújtanak hasznos segítséget.

Bár az angol szakirodalomban a FORTRAN nyelvnek az ALGOL-tól némileg eltérő terminológiája van, mi eléggé következetesen az ALGOL 60 szakkifejezéseihez ragaszkodtunk. Nem láttuk értelmét ugyanis annak, hogy csak azért, mert ALGOL helyett FORTRAN-ról van szó, ugyanazt a fogalmat másként nevezzük. Az ALGOL terminusait elég elterjedtnek és közismertnek tartjuk ahhoz, hogy ezekre támaszkodhassunk. (Így lett pl. a „name”-ból „azonosító”, a „dummy argument”-ből „formális paraméter”, az „array element”-ből „indexes változó” stb.) Ha valamely fogalom angol elnevezését ésszerű módon le tudtuk fordítani magyarra, akkor megtettük („common block” — „közös adatmező”), de ha nem, akkor meghagytuk az angol eredetit („file”).

A programnyelvi tanfolyamokon, de egyéb alkalmakkor is, gyakran találkoztunk azzal a kérdéssel, hogy a két „rivalizáló” programozási nyelv, az ALGOL 60 és a FORTRAN közül melyik a „jobb”? — Véleményünk szerint a kérdés ilyen feltevésének nincs értelme. A két nyelv közül adott esetben az a jobb, amelynek segítségével a konkrét feladatot hatékonyabban lehet megoldani. Ennek a hatékonyságnak pedig igen sok tényezője van: a programozó személyes jártassága egyik vagy másik nyelvben; a szűkebb tudományág vagy műszaki terület programozási szakirodalma (egyes témakörök inkább ALGOL-ban vannak kidolgozva, mások FORTRAN-ban); az alkalmazott algoritmus jellege; az adott gép ALGOL, ill. FORTRAN gépi reprezentációjának minősége stb. Általánosságban csak annyit mondhatunk, hogy bonyolultabb számítási eljárások programozása ALGOL-ban általában kevesebb írásmunkával jár; ezzel szemben FORTRAN-ban a be- és kivitel programozása kényelmesebb, és — ami sok esetben ugyancsak fontos — egységes, ugyanis a be- és kivitel a FORTRAN-ban része a hivatkozási nyelvnek.

A fejlődés természetesen a ma használt programozási nyelveknél sem áll meg. Minden bizonnyal néhány év múlva nagyobb teljesítményű gépek és újabb, kényelmesebb programozási nyelvek váltják fel a maiakat. A programozási ismereteket bővíteni és fejleszteni kell. Úgy gondoljuk azonban, hogy

a FORTRAN nyelv még néhány évig kielégíti a felhasználói igényeket.

Végezetül köszönetünket fejezzük ki *Gömbös Ervin* lektornak hasznos észrevételeiért, valamint *Harsányi Menyhértné*nek, *Haviár Erikának* és *Kerek Margitnak* a technikai munkákban nyújtott segítségükért.

A szerzők

Első fejezet

A PROGRAMOZÁS AUTOMATIZÁLÁSA ÉS A FORTRAN PROGRAMOZÁSI NYELV

1. Bevezetés

A számológépek rohamos fejlődésével és elterjedésével egyre nagyobb lehetőség nyílik arra, hogy a tudomány, a technika és a gazdasági élet számtalan feladata közül minél többet oldhassunk meg gépi úton. A numerikus analízis és a rokon tudományágak folyamatosan adnak olyan újabb számítási eljárásokat, amelyek segítségével a feladatmegoldás célszerűbben gépesíthető. Ugyanakkor egyre több szellemi energiát köt le a feladatok „gépre vitele", a *programozás* és *kódolás*. Ez a kettős folyamat jelenleg is tart, bár már régebben megindult a programozási munka megkönnyítése céljából a *programozási nyelvek* kidolgozása és továbbfejlesztése is.

Bizonyos fejlődési szintet képviselnek az ún. *algoritmikus* programozási nyelvek. Két fontosabb ok vezetett kialakulásukhoz: egyrészt az a felismerés, hogy a programozási munkában igen sok művelet automatizálható, azaz elvégeztethető magával a géppel, másrészt az az igény, hogy a matematikai formulákat közvetlenül — vagy legalábbis lehető csekély módosítással — be lehessen írni a programokba. Azokat az algoritmikus nyelveket, amelyek csak egy meghatározott számológép-típuson használhatók fel, *autokódoknak* szokás nevezni. Ilyen például a NATIONAL ELLIOT-803 gép ELLIOT autokódja, a MINSZK-22 gép MITRA nevű autokódja stb. Ezeken a nyelveken lényegesen könnyebb a programozás, mint a gép nyelvén, az ún. *gépi kódban.*

Szükség volt továbbá olyan algoritmikus nyelv kifejlesztésére, amelyen írt program a különféle számológépeken egyaránt felhasználható, azaz ebből a szempontból *univerzális.* Ez az igény vezetett a magasabb szintű nyelvek, a FORTRAN és az ALGOL kialakulásához.

Felismerték azt is, hogy sok területen a feladatok tipi-

kusak, azaz igen sok köztük a hasonlóság. Ezért célszerűnek látszott bizonyos közös vonásokat már a programozási nyelvekben figyelembe venni. Így alakították ki az ún. *probléma-orientált nyelveket*, mint például az adatfeldolgozási feladatokra kidolgozott COBOL nyelvet, a sorbanállási problémák vagy rendszerszimulációs feladatok programozására alkalmas SIMSCRIPT vagy GPSS nyelveket és egy sor további programozási célnyelvet. Ily módon a fejlődés az idők folyamán rendkívül sok „nyelvjárás" kialakulásához vezetett, amelyeknek még a felületes áttekintése is elég nehéz.

Legújabban olyan programozási nyelvek kidolgozása is megindult, melyek — legalábbis elvben — több nyelv előnyeit egyesítenék, illetve amelyek magukban foglalnának több programozási nyelvvel egyenértékű ún. „nyelv-modulokat". Ez a törekvés érezhető a PL 1 nyelven és részben az ALGOL 68 nyelvtervezeten is. (Egyes szerzők e két nyelvet tekintik univerzális nyelvnek.)

Az egyes programozási nyelvek közül egy konkrét számológép programozására csak azokat lehet használni, amelyhez a számológépnek van ún. *fordítóprogramja*. A fordítóprogram segítségével ui. a gép az algoritmikus nyelven megírt programot „lefordítja" a gép nyelvére, vagyis megszerkeszti a futtatásra, azaz a kívánt feladat gépi megoldására alkalmas *gépi kódú programot*. Emellett a fordítóprogram hatáskörébe utalhatók bizonyos hibajelzési feladatok is. A továbbiakban a futtatásra alkalmas programot *célprogramnak*, vagy *tárgyprogramnak*, a megfelelő programozási nyelven írt programot pedig *forrásprogramnak* nevezzük.

A programozó számára nem okvetlenül szükséges, hogy ismerje a számológép gépi nyelvét, az ún. *belső kódot*, vagy éppen a fordítóprogram működését. Elég, ha ismeri a programozási nyelv — a továbbiakban *forrásnyelv* — szabályait. Ezzel a megoldandó feladatot be tudja programozni. A programozó számára a számológép a megfelelő fordítóprogrammal együtt egységes egészet képez.

Mindez igaz a programozási nyelvekre általában, így a FORTRAN nyelvre is.

2. A FORTRAN nyelvek kialakulása

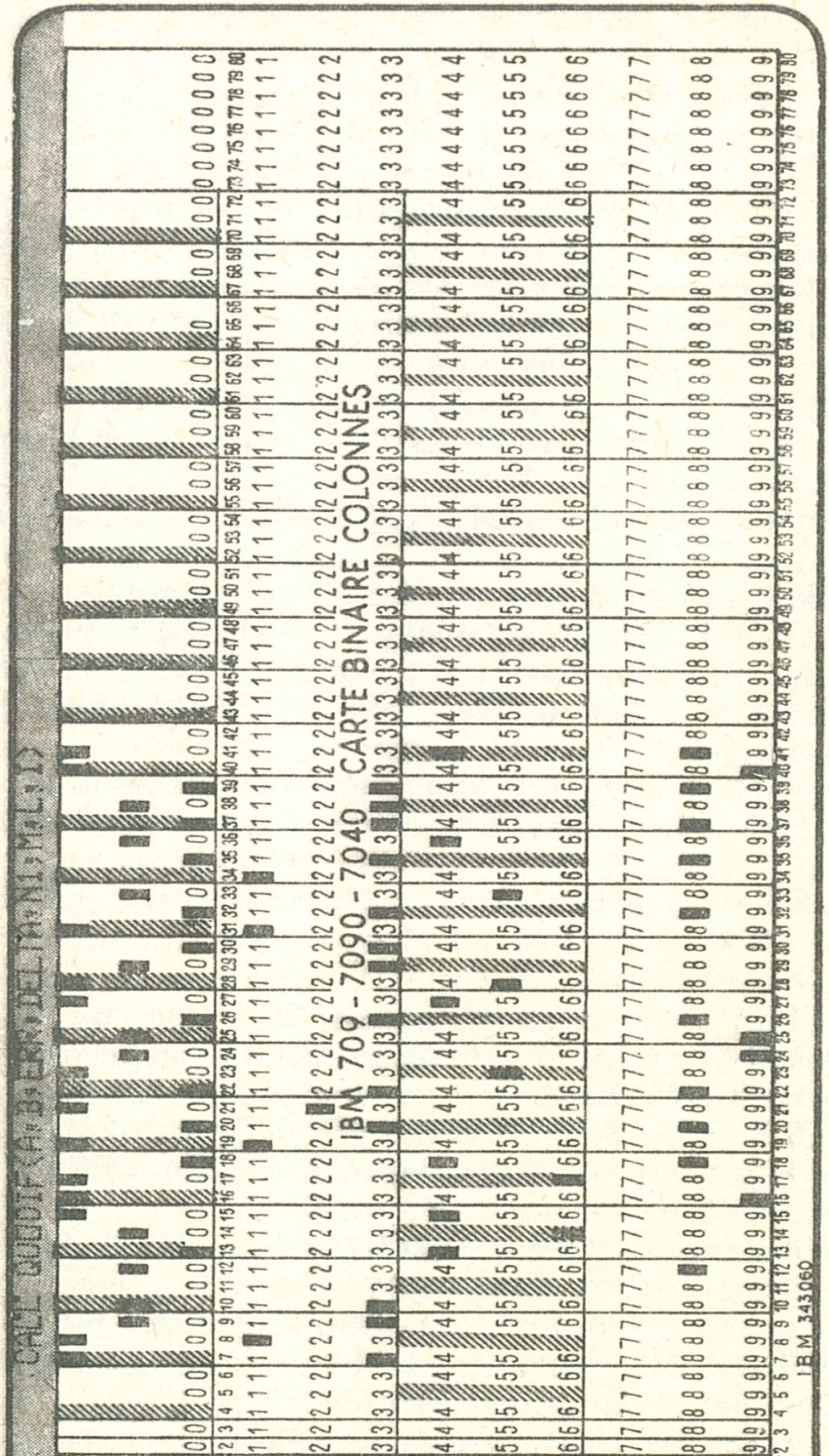
Könyvünkben az egyik legkorábban kifejlesztett univerzális algoritmikus nyelvcsaláddal, a FORTRAN nyelvekkel foglalkozunk.

A FORTRAN nyelveket úgy tekinthetjük, mint az IBM Vállalatnak (International Bussiness Machines Corp.) és leányvállalatainak gépeire írt autokódok továbbfejlesztett változatait. A legelső említésre méltó változat a FORTRAN-II, melyet az IBM cég 1956-ra dolgozott ki. Később a nagyobb IBM gépeket egy bővebb algoritmikus nyelven, a FORTRAN-IV nyelven írt programok lefordítására alkalmas fordítóprogrammal is ellátták.

1962 májusában lehetővé vált, hogy az Amerikai Szabványügyi Hivatal (American Standards Association, rövidítve: ASA) munkacsoportot hívjon létre, amelynek feladatául tűzte ki, hogy a FORTRAN nyelveket *szabványos* programozási nyelvként definiálja. A szabványtervezet 1964 októberére készült el, és azóta is ezt tekintik a FORTRAN nyelv hivatalos definíciójának [1].

A szabványtervezet a FORTRAN-II és FORTRAN-IV nyelveknek megfelelően két változatot definiál oly módon, hogy a több programozási lehetőséget nyújtó nyelv (a továbbiakban ASA FORTRAN) „részhalmazaként" értelmezi a szűkebb, ún. BASIC FORTRAN nyelvet. Ez azt jelenti, hogy minden BASIC FORTRAN nyelven megírt program egyben ASA FORTRAN-ban írt program is, és jelentése ugyanaz, akár ASA FORTRAN-ban, akár BASIC FORTRAN-ban értelmezzük. Az ASA FORTRAN fordítóprogramjai tehát tökéletesen lefordítják a BASIC FORTRAN nyelven írt programokat is. Az ASA FORTRAN bizonyos utasításai viszont a BASIC FORTRAN-ban hiányoznak.

Könyvünkben az ASA FORTRAN nyelvvel foglalkozunk és ezentúl ezt nevezzük FORTRAN nyelvnek vagy *hivatkozási nyelvnek*. Többször utalunk azonban ún. FORTRAN *gépi reprezentációkra* is. Ezekről azért kell beszélnünk, mert az egyes számológépekhez a gép adottságaitól függően olyan FORTRAN fordítóprogramokat írtak, amelyek néhány programozási kérdésben több, néha — ritkábban — kevesebb szabadságot, illetve lehetőséget biztosítanak a programozó számára, mint a


IBM 709 - 7090 - 7040 CARTE BINAIRE COLONNES
IBM 343060

szabványos ASA FORTRAN nyelv. Ezért a gyakorlatban a programozási feladat kapcsán mindig figyelemmel kell lennünk a gépi reprezentációk sajátosságaira.

A FORTRAN nyelveken érezhető egy speciális gyakorlati körülmény hatása is. Az IBM cég előszeretettel használ lyukkártyát adatok és programok bevitelére (lásd az 1. ábrát). Ennek következtében a FORTRAN nyelvnek is sajátossága az „egy sor—egy utasítás" megfeleltetés. Pontosabban: az egyes utasításokat a FORTRAN-ban mindig csak új sorban, a 7. karakterpozíciótól kezdve lehet elkezdeni. Az ilyenfajta formai követelmények speciális programozói előnyökkel és hátrányokkal járnak.

3. Alapfogalmak

Egy FORTRAN nyelven írt programnak — ahhoz, hogy értelmes legyen — ki kell elégítenie bizonyos formai („nyelvtani") követelményeket. Azt mondjuk, hogy *szintaktikusan* (vagy *formálisan*) *helyes* az olyan program, amely ezen „nyelvtani" szabályoknak eleget tesz. A fordítóprogramoktól megköveteljük, hogy szintaktikusan helyes forrásprogramok alapján végrehajtható célprogramokat készítsenek, a szintaktikusan hibás forrásprogramokról pedig közöljék ezek hibás voltát, és adjanak bizonyos felvilágosítást a hibák jellegéről.

A FORTRAN nyelven írt program utasítások sorozatából áll. Az egyes FORTRAN utasítások felírásának legáltalánosabb formai szabályai a következők:

1. A FORTRAN utasításokat *sorokba* írjuk, minden utasítást új sorba. Minden sorban 80 *karakterpozíció* van, ezek közül utasításokat csak a 7.-től a 72.-ig szabad elhelyezni. Egy karakterpozícióban a FORTRAN *jelkészlet* egyetlen jele — ún. *karaktere* (pl. betű, számjegy stb. lásd alább) — foglalhat helyet. A pozíciókat balról jobbra számozzuk.

2. Ha egy utasítás nem fér el egy sorban, lemaradó részeit folytatólagosan kell az ún. *folytatósorokban* elhelyezni, a 7.-től a 72. karakterpozícióig. Az egymást követő folytatósorok száma maximálisan 19, tehát egy utasítás maximálisan 20 sorra terjedhet.

3. A folytatósorok 6. karakterpozíciójában 0-tól és az ún. *betűköz (szóköz, space)* karaktertől különböző karakter, rend-

szerint egyjegyű egész szám áll. Ez használható a folytatósorok sorszámozására. Szükséges, hogy az első karakterpozíción *ne legyen* C betű.

A folytatósorok első 5 karakterpozícióján nem lehet hasznos információ, ezért sok gépi reprezentáció előírja, hogy ezek a pozíciók üresek maradjanak.

4. Ha a sor első karakterpozícióján C betű áll, akkor ez a sor nem tartozik a programhoz; a fordítóprogram nem is fordítja le. Az ilyen sorokban olyan közleményeket, magyarázatokat lehet elhelyezni, amelyeket a programozó a program olvasójának szán („comment sor"). Ilyen típusú sorokat nem szabad folytatósorokban folytatni.

5. Az első 5 karakterpozíció ún. *címkék* elhelyezésére szolgál. Címke legfeljebb ötjegyű, előjeltelen egész szám lehet. Szám elején álló nulla betűköz karakternek számít, így a 0 nem lehet címke.

Annak érdekében, hogy ezeknek a formai szabályoknak a betartása ne okozzon gondot a programozónak, a programokat *programlapokra* szokás írni* (lásd az 1. mellékletet). A programlapok alapján készülhetnek el az egyes programkártyák (minden sorhoz egy-egy kártya tartozik). A kártyákon az egy oszlopban levő lyukak kombinációja egy karakternek felel meg, melyet a könnyebb kezelhetőség kedvéért a kártya felső szélére általában rá is gépelnek. Könnyen lehet az egyes kártyákat cserélni vagy további kártyákat közéjük beiktatni stb., így a program könnyen javítható, fejleszthető, módosítható.

Ahhoz, hogy programunk helyesen működjék, nem elég, ha szintaktikusan helyes. Magától értetődően szükséges az is, hogy éppen azokat a műveleteket hajtsa végre, amelyeket a programozó kívánt. Az ilyen értelemben is helyes programokat *szemantikusan* helyesnek nevezzük. Világos, hogy egy program lehet szintaktikusan hibátlan, de szemantikailag hibás. Az ilyen programot a fordítóprogram lefordítja, de a lefordított program vagy egyáltalában nem, vagy hibásan működik.

Az egyes utasításokat csakis az ún. FORTRAN jelkészlet karaktereiből építhetjük fel. Ezek:

* A programlapok kitöltésekor ügyeljünk arra, hogy a nulla számjegyet és a nagy O betűt jól meg lehessen különböztetni. Például utóbbiakat húzzuk át vízszintesen úgy, ahogyan az 1. mellékletben is látható.

1. az angol ábécé nagybetűi: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z.

2. a számjegyek: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

3. az 1. táblázat szerinti speciális karakterek.

1. táblázat

A FORTRAN nyelv speciális jelei

Jel	A jel neve
▽	betűköz[a)]
=	egyenlőségjel
+	plusz
−	mínusz
*	szorzásjel
/	törtvonal
(	kezdő zárójel
)	végzárójel
,	vessző
.	pont
$	dollárjel[b)]

a) A ▽ jel a betűközöket szimbolizálja. Olyankor tesszük ki, amikor valamely okból jól láthatóan fel kívánjuk tüntetni a betűközök számát.
b) A dollárjelet, vagy árfolyamjelet nem minden gépi reprezentáció használja, ezért sokszor nem tekintik a FORTRAN jelkészlethez tartozónak.

A FORTRAN utasítások nagy részét arra használják, hogy velük bizonyos matematikai mennyiségeknek, a változóknak, értéket adjanak. Ezek az ún. *értékadó utasítások.* Például a*

```
|19    X = X-Y+12.5
```
(1.1)

értékadó utasítás az X változó korábbi értéke alapján kiszámíttat egy új X értéket, amely az egyenlőségjel jobb oldalán levő kifejezés számértékével, ún. *aktuális értékével* egyezik meg.

* Tekintettel arra, hogy a FORTRAN nyelv nem tizedesvesszőt, hanem tizedespontot használ, az egyszerűség kedvéért a továbbiakban mindig ezt követjük. Az egyes utasítások helyzetét úgy jelöljük, hogy a sor elejére – tehát az első karakterpozíció elé – egy függőleges vonalat húzunk, vagy pedig a folytatósorra utaló – a hatodik – karakterpozíciót két függőleges vonallal jelezzük.

A gép X előző értéke helyébe az utasítással meghatározott új értéket írja be. Például ha az utasítás végrehajtása előtt X pillanatnyi értéke 3.5, Y pillanatnyi értéke pedig 2.5 volt, akkor az utasítás végrehajtása után X értéke 13.5 lesz (Y értéke továbbra is 2.5 marad). Ugyanaz a változó egy programban többször is kaphat új értéket, akár értékadó utasítások, akár a *perifériális egységek* segítségével végzett *beolvasás* hatására.

Az (1.1) utasítás előtt álló „19" szám az utasítás *címkéje*. A cimkék teszik lehetővé, hogy a program egyes utasításai egymásra hivatkozhassanak. Vannak olyan utasítások, ún. *vezérlésátadó utasítások*, amelyek hatására a gép a soron következő utasítás helyett a megadott címkéjű utasítást hajtja végre. A vezérlésátadás függhet bizonyos feltételek teljesülésétől is. Eszerint beszélünk *feltételes*, illetőleg *feltétlen* vezérlésátadó utasításról. A címke nélküli

```
  | |GO TO 19                                  (1.2)
```

illetve a címkével ellátott

```
12 | |GO TO 19                                 (1.3)
```

feltétlen verzérlésátadó utasítások hatására a vezérlés az (1.1) utasításra adódik át. Látjuk, hogy a vezérlésátadó utasításnak is lehet címkéje.

Nagyon fontos fogalom az *azonosító* fogalma. Ezen körülbelül ugyanazt értjük, mint a „név" köznapi fogalmán. Azonosítója van többek között minden változónak. Például az (1.1) utasításban az X betű egy változót azonosít vagy jelöl; azaz megkülönbözteti a többi változótól. Azonosítója van másféle programozási objektumoknak (ún. *szintaktikus egységeknek*) is: vektoroknak, mátrixoknak, szegmenseknek (lásd a 4. fejezetet), sőt (gépi reprezentációkban) teljes programoknak is.

A FORTRAN fordítóprogramnak sok esetben szüksége van arra, hogy a fordítás folyamán a programban előforduló azonosítókról bizonyos felvilágosításokat, információkat kapjon. Ilyen információ például, hogy az azonosító vektort jelöl; hogy a vektornak 5 komponense van stb. Ezeket az információkat ún. *deklaratív*, vagy *nem végrehajtható* utasítások segítségével adhatjuk meg. A FORTRAN nyelv sajátsága, hogy ha egyes azonosítókat nem deklarálunk, akkor a fordítóprogram meg-

felelő szabályok alapján saját maga definiálja azok tulajdonságait. Ez megkönnyíti a program írását, bár olykor hibához is vezet.

A deklaratív utasításokat a 4. fejezetben tárgyaljuk.

Nagyon fontos szabálya a FORTRAN nyelven írt forrásprogramoknak, hogy az utasításokban tetszőleges számban és helyen lehet jelentés nélküli betűköz karaktereket elhelyezni. Ez a szabály még a több karakterből álló azonosítókra is érvényes: betűközök helyezhetők az azonosítók betűi közé is, anélkül, hogy az azonosítók jelentése megváltozna. Például a következő utasítások jelentése teljesen megegyezik:

```
12  | | ALLAT = KUTYA+MACSKA1      (1.4)
1 2 | |AL LAT = KU TYA+MA CSKA1   (1.5)
```

Megjegyezzük, hogy egyes gépi reprezentációk nem engedik meg, hogy cimkék belsejében betűköz legyen.

Az 1. mellékleten egy FORTRAN példa-programot látunk. Sorravesszük most az egyes utasítások jelentését. (A sorokat kivételesen megszámoztuk az utolsó karakterpozíciókon.)

1. sor: A sor elején álló MASTER szó sok gépi reprezentációban azt jelenti, hogy az illető sor a program ún. fő szegmensének (lásd 5. fejezetet) kezdő sora. Utána egy szegmensazonosítót (TABELLAZAS) találunk.

2. sor: az 1.2 értéket adja az X változónak. Vegyük észre, hogy a tizedespont karakter leírásához is külön pozícióra van szükség.

3. sor: a 3.6 értéket adja az Y változónak. Ez az utasítás a 18-as címkével van ellátva.*

4. sor: értékadó utasítás, mely X és Y aktuális értékéből kiszámítja F értékét. Az egyenlőségjel jobb oldalán egy képlet, ún. *aritmetikai kifejezés* áll. A „*" (csillag) szorzást jelent, két egymás utáni csillag pedig hatványozást. A számok felírási módjaival a 2. fejezetben ismerkedünk meg. A képlet a szokásos írásmódban a következő alakú:

$$f = \frac{1}{\sqrt[3]{1+2x^2+3y^2}} \qquad (1.6)$$

* Az 1. mellékleten bemutatjuk a cimkék lehetséges elhelyezési módjait is: az első 5 karakterpozícióra bárhová írható a címke; a hivatkozási nyelv azt is megengedi, hogy belsejében betűköz legyen!

Az utasítás a 15-ös címkével van ellátva.

5. sor: a WRITE kiíró (nyomtató) utasítás, amely X, Y és F értékét íratja ki. Lásd a Hatodik fejezet 2. pontját.
6. sor: FORMAT utasítás. A nyomtatási képet határozza meg. Tárgyalását lásd a 6. fejezetben.
7. sor: értékadó utasítás, mely Y értékét megnöveli egy tizeddel.
8. sor: ún. aritmetikai IF utasítás. Attól függően, hogy a zárójelben negatív, nulla vagy pozitív aktuális értékű aritmetikai kifejezés áll, a vezérlés az utasítás végén felsorolt három cimke valamelyikére adódik át. Nevezetesen, példánkban

 ha $Y < 36.01$, akkor a 15-ös címkére,
 ha $Y \geqq 36.01$, akkor pedig a 16-os címkére.
9. sor: értékadó utasítás, amely X értékét megnöveli egy tizeddel. A 16-os címkével van ellátva.
10. sor: a 8. sorhoz hasonló aritmetikai IF utasítás
11. sor: STOP, vagyis állj utasítás.
12. sor: a szegmens végét jelző END utasítás.

Ha végigkövetjük a program működését, láthatjuk, hogy az az (1.6) formulával megadott függvényt tabellázza az $1.2 \leqq X \leqq 12$; $3.6 \leqq Y \leqq 36$ intervallumban, mindkét változóban tizedenként.

A könyv további fejezeteiben részletesen tárgyaljuk majd azokat a szabályokat, amelyeket egy ilyen program megszerkesztésekor figyelembe kell vennünk. E helyen célunk csupán a FORTRAN nyelv néhány jellegzetességének előzetes illusztrálása volt. Futólag megismerkedtünk az értékadó, a vezérlésátadó és kinyomtató utasításokkal, a programírás formátumával és még néhány formai és tartalmi szabállyal. A fenti — kissé ötletszerű — tárgyalás után, a következő fejezettől kezdve áttérünk a nyelv rendszeres ismertetésére.

ARITMETIKAI ÉS LOGIKAI KIFEJEZÉSEK. RELÁCIÓK

1. Értéktípusok

A FORTRAN nyelv igen sok utasítása használ fel aritmetikai és logikai kifejezéseket, valamint relációkat. Célszerű tehát e fogalmakkal az utasítások tárgyalása előtt részletesen megismerkednünk.

Tegyük fel, hogy sikerült valamilyen mennyiséget ismert mennyiségekkel kifejeznünk. Például az $ax^2+bx+c=0$ másodfokú egyenlet egyik gyökét kifejezhetjük az

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a} \tag{2.1}$$

képlettel. Ha az egyenlőségjel jobb oldalán álló kifejezést — a FORTRAN nyelv később ismertetendő szabályainak megfelelően — leírjuk, például a következő módon:

(−B+(B**2−4.0*A*C)**0.5)/(2.0*A) (2.2)

akkor *aritmetikai kifejezéshez* jutottunk. Ha ismeretes A, B és C értéke, akkor a (2.2) aritmetikai kifejezést „ki lehet értékelni”, vagyis van *aktuális értéke.* Ha viszont A, B és C ismeretlenek, vagy a program még nem adott nekik értéket, akkor nincs értelme a kifejezés aktuális értékéről beszélni. A továbbiakban mindig feltesszük, hogy kifejezéseink olyan programba vannak beágyazva, mely az előforduló változóknak már előzőleg adott értéket.

Mivel egy aktuális érték a számológépben többféleképpen „ábrázolható”, a FORTRAN nyelv is az értékek többféle típusát vezeti be. Aritmetikai kifejezések aktuális értéke négyféle típusú lehet. Ezek közül az INTEGER vagy *„egész”* típusú értékeket a számológép pontosan ábrázolja. Ezek az értékek a pozitív és negatív egész számok és a nulla, a számológép-

ben ábrázolható értéktartományban. A REAL vagy „*valós*" típusú értékek csak bizonyos relatív pontossággal ábrázolhatók, mégpedig a gép típusától függően kb. 12—16 decimális jeggyel. A DOUBLE PRECISION vagy „*kétszeres pontosságú*" értékek csak a számábrázolás nagyobb relatív pontosságában térnek el a „*valós*" értékektől; a hivatkozási nyelv csak ennyit ír elő. Az ICT gépi reprezentációban ez azt jelenti, hogy a pontosság 21 tizedes jegy, szemben a REAL értékek 12 jegy pontosságával. A COMPLEX vagy „*komplex*" típusú értéket a gépben egy „valós" típusú számpár képviseli.

Ha például két REAL típusú értéket összehasonlítunk (relációba hozunk), akkor van értelme egy olyan állításnak, hogy az első érték nagyobb a másodiknál, pl.: A nagyobb, mint B. Ez a megállapításunk A és B adott értékei mellett lehet *igaz* vagy *hamis*. Ilyenformán relációnk — A és B aktuális értékétől függően — felveszi az igaz, vagy a hamis értéket. Ez nem számérték, hanem új értéktípus, ún. LOGICAL vagy „*logikai*" érték. Nyilvánvaló, hogy logikai érték csak kétféle lehet: igaz (TRUE), vagy hamis (FALSE).

Felírhatók olyan ún. *logikai kifejezések*, amelyek „kiértékeléséhez" logikai értékekkel kell speciális, ún. *logikai műveleteket* végezni. Sok hasonlóság van az aritmetikai és logikai kifejezések felírása és kiértékelése között, például a zárójelek használata tekintetében is.

Most sorravesszük azokat az alapvető szintaktikus egységeket, amelyeket kifejezések felírásában használunk.

2. Számok

A négyféle típusú számértéknek — INTEGER, REAL, DOUBLE PRECISION és COMPLEX — megfelelően, a FORTRAN-ban négyféle típusú *számról*, vagy *számkonstansról* beszélhetünk.

Egész típusú (INTEGER) számot egy vagy több számjeggyel írhatunk fel; a legelső számjegyet egy előjel is megelőzheti. A pozitív előjel elhagyható. A számok első jegyei értéktelen nullák is lehetnek. Helyesen vannak felírva például a következő egész számok:

1 +1 0023 −75 1969

Mivel a programsorokban tetszőleges pozícióban elhelyezhetünk betűköz karaktereket, amelyek nem érintik a program jelentését, a számok belsejében levő üres karaktereknek sincs semmiféle jelentőségük. Így az

1 +1 0 02 3 −7 5 1 969

számok megegyeznek a fentiekkel.

Ez a megjegyzés mindegyik számtípusra érvényes.

A gépi reprezentációkban a különféle típusú számok abszolút értéke nem haladhatja meg a megfelelő gép számábrázolási rendszeréből eredő korlátokat.

A *valós típusú* (REAL) számok öt részből állhatnak: *előjelből, egész részből, tizedespontból, törtrészből* és *kitevőrészből.* Például a $-25.7117 \cdot 10^9$ számértéket a következőképpen írhatjuk fel:

−	25	.	7117	E+09	
↓	↓	↓	↓	↓	
előjel	egész rész	tizedes-pont	tört-rész	kitevő-rész	(2.3)

Ennél a felírási példánál mind az öt részt felhasználtuk. Ezek egyike-másika bizonyos megszorításokkal elhagyható. Az erre vonatkozó szabályok a következők:

1. Az előjeltelen valós szám pozitív.
2. Valós számban a tizedespont és a kitevőrész közül legalább az egyiknek szerepelnie kell.
3. Valós számban az egész rész és a törtrész közül legalább az egyiknek szerepelnie kell.

Az E betű hozzátartozik a kitevőrészhez, annak első jele.

Példák helyesen felírt valós számokra:

−11 500. +40.007 072 .5147

−115.E2 4.0007E+1 .05E−12

+6E13 −08.80E−03 7.2E−00

Példák hibásan felírt valós számokra:

+0,5 E−3 +.E+3

A számológépek nagyrészt bináris számrendszerben dolgoznak, ezért a felírt REAL típusú számokat általában nem lehet pontos értékükkel ábrázolni a gépen. Ennek következtében bevitelkor kerekítésre kerül sor.

A *kétszeres pontosságú* (DOUBLE PRECISION) számokat a valós típusúakhoz hasonló módon írjuk fel, csupán a kitevőrész E betűje helyett D betűt használunk. A REAL típustól eltérően azonban itt *mindig szerepelnie kell a kitevőrésznek* és evvel együtt a D betűnek.

Példák helyesen felírt dupla pontosságú számokra:

−115.D2 4.0007D+1 .05D−12

+6D13 3.14159265358979323846D00

Példák hibásan felírt dupla pontosságú számokra:

−0,47D+5 D−06 +.D+2 0.52134827411827

Magától értetődően a 4.0007E+1 valós, és a 4.0007D+1 kétszeres pontosságú szám belső gépi ábrázolásban egymástól eltér,* ezért számértékeik is különböznek.

Komplex típusú (COMPLEX) számokat valós számpárok segítségével írhatunk fel. A számpárt zárójelek közé kell zárni; a valós rész után vessző kerül, majd a képzetes rész következik. Például:

(0.1, −0.1732) (−0.14142E1, +1.4142)

Az ASA FORTRAN hivatkozási nyelv nem határozza meg sem a számok abszolút értékének maximumát, sem pontosságát. Ezek a gépi reprezentációktól függenek. Előírja azonban, hogy a kétszeres pontosságú ábrázolás relatív pontosságának meg kell haladnia a valós számok ábrázolásának relatív pontosságát.

3. Változók, azonosítók

Ahhoz, hogy aritmetikai kifejezéseket felírhassunk, jelölnünk kell a bennük szereplő változókat. A (2.2) formulában ezt egy-egy nagybetűvel tettük. Tekintve, hogy mindössze 26 nagybetűt használhatunk, a jelölésre egy karakter nyilván általában

* A tízes számrendszerben a megadott két szám pontosan egyenlő. A gép viszont kettes számrendszerben dolgozik, amiben ezeket a számokat csak végtelen szakaszos „bináris törttel" (vö. végtelen szakaszos tizedestört!) lehet megadni. De a gép csak véges sok számjegyet jegyez fel; valós „számábrázolás" esetén kevesebb jegyet vesz figyelembe, mint kétszeres pontosságú esetén. Így ez a két „közelítő érték" vagy „számábrázolás" egymással általában érték szempontjából sem egyenlő.

lában nem elegendő. A FORTRAN nyelv megengedi, hogy erre a célra több karakterből álló karaktersorozatokat, — ún. *azonosítókat* — használhassunk. (Hasonló azonosítókkal lehet vektort, mátrixot, szegmenst vagy programot is jelölni.)

Az ASA FORTRAN-ban az azonosítók maximális hossza hat karakter. Az első karakter csak betű lehet, a többi betű vagy számjegy. Nem ajánlatos a más célra fenntartott karaktersorozatokat — az ún. *alapszavakat* — (GOTO, WRITE, ...) vagy ilyenekkel kezdődő karaktersorozatokat (GOTO 12, IFJU, ...) azonosítóként használni. A betűköz karakter nem tartozik az azonosítóhoz, még ha belsejében is van. Például a következő azonosítók megegyeznek:

AZUREG AZ UR EG AZUR EG AZ UREG

Szabad azonosítót folytatósorban folytatni. Például:

```
| |AZUREG = ALUK*VOROS+AZU
|1|REG/12.0
```

Ebben a felírásban a második sor elején álló 1 számjegy nem tartozik az AZUREG azonosítóhoz, hanem a folytatósort jelzi!

Egyes gépi reprezentációk hosszabb azonosítókat is megengednek. A programozási gyakorlatban viszont lehetőleg elkerülik az azonosítók és az alapszavak elválasztását vagy tagolását.

A FORTRAN nyelv aritmetikai kifejezésekben egész, valós, kétszeres pontosságú és komplex típusú változókat használ, melyek a megfelelő típusú aktuális értékeket vehetik fel. A *logikai típusú* (LOGICAL) változókat logikai kifejezésekben használhatjuk. Ezek a TRUE és FALSE logikai értékeket vehetik fel. Velük részletesen a Második fejezet 7. pontjában foglalkozunk.

Hacsak valamilyen deklaratív utasítással (lásd 4. fejezet) eltérő előírást nem adunk meg, akkor a FORTRAN programokban *az* I, J, K, L, M *és* N *betűkkel kezdődő azonosítókat egész típusú, a többi betűvel kezdődőeket valós mennyiségek, változók jelölésére használjuk*. (Egész típusú mennyiség lehet például olyan vektor is, melynek komponensei egész típusú számok.) Ezt a szabályt a későbbiekben az *automatikus típusdeklarálás* szabályának fogjuk nevezni.

Vektorok és mátrixok egyes komponenseit *indexes változók-*

kal jelöljük. Ilyenkor pl. a vektor azonosítóját zárójelben követi a komponens sorszáma (az index). Így például a

VEKTOR (3)

indexes változó a VEKTOR azonosítójú vektor harmadik komponensét jelenti. Az indexes változók használatától egyelőre eltekintünk.

4. Aritmetikai műveletek

Az egyes aritmetikai műveletek jelölésére a FORTRAN-ban a következő *műveleti jeleket* használjuk:

1. Összeadás. Jele: +
2. Kivonás. Jele: –
3. Szorzás. Jele: * (csillag)
4. Osztás. Jele: /
5. Hatványozás. Jele: **

Az aritmetikai műveletek nincsenek értelmezve tetszőleges típusú mennyiségek (operandusok) között, a művelet eredményének típusát pedig az operandusok típusa határozza meg. A négy alapműveletre ezeket a 2. táblázat mutatja:

2. táblázat

Az alapműveletek eredményének típusai

OPERANDUSOKTÍPUSA	INTEGER	REAL	DOUBLE PRECISION	COMPLEX
INTEGER	INTEGER	–(R)	–(D)	–(C)
REAL	–(R)	REAL	DOUBLE PRECISION	COMPLEX
DOUBLE PRECISION	–(D)	DOUBLE PRECISION	DOUBLE PRECISION	–
COMPLEX	–(C)	COMPLEX	–	COMPLEX

Megjegyzések:

1. A „ – ” jelek tiltott kombinációkat jelentenek.

2. A zárójelbe tett típus-kezdőbetűk azt jelölik, hogy egyes gépi reprezentációkban milyen típusú eredményt rendelnek az adott tiltott kombinációkhoz.

3. Ha egészet osztunk egésszel, akkor az ún. „egészek osztásának" megfelelő eredményt kapjuk, amelynek értéke az a legnagyobb abszolút értékű egész szám, melynek előjele megegyezik a pontos eredmény előjelével, és abszolút értéke nem haladja meg a pontos eredmény abszolút értékét. Például

$$13/4=3 \quad \text{és} \quad 15/(-4)=-3$$

4. A nullával való osztás nincs értelmezve.

A hatványozásra az operandusok és az eredmény lehetséges típusait a 3. táblázat tartalmazza:

3. táblázat

A hatványozás eredményének típusa

Az alap típusa	A kitevő típusa			
	INTEGER	REAL (1)	DOUBLE PRECISION (1)	COMPLEX
INTEGER	INTEGER (2)	–(R) (3)	–(D) (3)	–
REAL	REAL	REAL	DOUBLE PRECISION	–
DOUBLE PRECISION	DOUBLE PRECISION	DOUBLE PRECISION	DOUBLE PRECISION	–
COMPLEX	COMPLEX	–	–	–

Az argumentumok aktuális értéke nem befolyásolja az eredmény típusát, de elvégezhetetlenné teheti a hatványozást. Így:

1. Negatív mennyiség nem emelhető valós vagy kétszeres pontosságú kitevőre. A zérus is csak pozitív egész kitevőre emelhető.

2. Egész számot negatív egész kitevőre emelve általában nullát kapunk eredményül, az egész osztáshoz hasonló kerekítési szabály miatt.

3. Egyéb jelöléseink megegyeznek a 2. táblázatban alkalmazottakkal.

Az ALGOL nyelvtől eltérően tehát a hatványozás eredményének típusa nem függ az operandusok aktuális értékétől.

Látható, hogy a hatványozás jele (az egymás utáni két csillag) két szorzásjel egymás után. Tehát ha szorzást akarunk programozni és — a nyomdai szokásoknak megfelelően — folytatósor elején megismételjük az előző sor végére tett szorzásjelet, a képlet szemantikailag hibás lesz, mert szorzás helyett hatványozást ír elő. Ügyeljünk arra, hogy programunkban két aritmetikai műveleti jel semmilyen módon ne kerüljön egymás mellé! Ez — a szorzásjel ismétlésének kivételével — szintaktikus hiba.

5. Aritmetikai kifejezések

A legegyszerűbb aritmetikai kifejezéseket számok, azonosítók (továbbiakban együttesen: operandusok) és aritmetikai műveleti jelek segítségével írhatjuk fel. (Az aritmetikai kifejezések fogalmát később az indexes változók, a standard függvények, valamint a zárójelek bevezetésével bővítjük.) Az ilyen legegyszerűbb aritmetikai kifejezések aritmetikai műveleti jelek és operandusok váltakozó sorozatai. Operandus most tehát számkonstans vagy változó-azonosító lehet. Példák ilyen egyszerű aritmetikai kifejezésekre:

```
I/J+K/N                      (2.4)
+273.16+Q/F                  (2.5)
—F**3.0+1.0—X**2/2.0*T       (2.6)
```

A gép e jelsorozatokat bizonyos értelmezési szabályoknak megfelelően értékeli ki. A közönséges matematikai képletírásnak megfelelően a műveleti jeleket ún. *elsőbbségük (precedenciájuk)* szerint csoportokba soroljuk:

1. *Elsődleges* műveleti jel a hatványozás jele: **
2. *Másodlagos* műveleti jelek a szorzás és osztás jelei: *, /
3. *Harmadlagos* műveleti jelek az összeadás és kivonás jelei: +, —

A kifejezéseket a bennük előforduló műveleti jelek alapján célszerű külön névvel ellátni. Ezek a nevek és a megfelelő szintaktikus egységek a következők:

1. *Elsődleges aritmetikai kifejezések* az operandusok, tehát olyan aritmetikai kifejezések, amelyekben nincsen műveleti jel.

2. A *tényező* vagy egyetlen elsődleges aritmetikai kifejezés, vagy két elsődleges aritmetikai kifejezés, a hatványozás jelével összekapcsolva. Tehát a tényező olyan aritmetikai kifejezés, amely nem tartalmaz sem másodlagos, sem harmadlagos műveleti jelet.

3. A *tag* egy vagy több tényezőből áll és a szomszédos tényezőket másodlagos műveleti jelek kapcsolják össze. Tehát a tag olyan aritmetikai kifejezés, melyben nem fordul elő harmadlagos műveleti jel.

Ezek szerint az elsődleges aritmetikai kifejezés egyben tényező és tag is. Példák:

Elsődleges kifejezések: Q; F; X; T; Z; U.

Tényezők: Q**F; X; T**2; Z; U.

Tagok: Q**F/X; T**2*Z; Q**F; U.

A tényező fenti definíciójának megfelelően a FORTRAN nyelvben tilos egy elsődleges kifejezést két hatványozási jel közé zárni, azaz aritmetikai kifejezésekben *tilos* az E**F**G típusú jelsorozatok leírása. (Ezt a szabályt zárójelek alkalmazásával lehet megkerülni, amit később látni fogunk.)

Megjegyezzük, hogy ezzel szemben az ALGOL nyelv megengedi a fenti szerkezetek felírását, hiszen balról jobbra szabálya meghatározza a két hatványozás elvégzésének sorrendjét. Ez az út a FORTRAN-ban nem járható!

A FORTRAN-ban az osztásjel (/) értelmezése a következő: az osztás jelét közvetlenül követő *egyetlen tényező reciprok értékét* kell képezni és az osztást ezzel való szorzással kell

helyettesíteni. (Ez megfelel az osztás szokásos értelmezésének.) Például:

$$A*B*C/D \quad \text{értelme} \quad A*B*C*\frac{1}{D}, \quad \text{azaz} \quad \frac{abc}{d};$$

$$A*B/C*D \quad \text{értelme} \quad A*B*\frac{1}{C}*D, \quad \text{azaz} \quad \frac{abd}{c};$$

$$A*B/C/D \quad \text{értelme} \quad A*B*\frac{1}{C}*\frac{1}{D}, \quad \text{azaz} \quad \frac{ab}{cd};$$

$$A/B/C*D \quad \text{értelme} \quad A*\frac{1}{B}*\frac{1}{C}*D, \quad \text{azaz} \quad \frac{ad}{bc}.$$

A tagok kiértékelése során ez a szabály matematikailag ugyanarra az eredményre vezet, mint az ALGOL nyelvben kimondott balról jobbra szabály. Megjegyezzük azonban, hogy ez nem feltétlenül jelenti azt, hogy numerikusan is egyező eredményt kapunk, hiszen általánosan ismert tény, hogy matematikailag egyenlő mennyiségek különböző numerikus számítás során gyakran eltérőnek adódnak. Erre később példát is látunk majd.

A FORTRAN nyelvben a fenti legegyszerűbb aritmetikai kifejezések kiértékelésének szabályai a következők:

1. *Precedenciaszabály.* Aritmetikai kifejezésekben az aritmetikai műveletek precedenciájuk sorrendjében végezendők el, hacsak zárójelek alkalmazásával más sorrend nincs előírva.

2. Egyenlő precedenciájú műveletek sorrendje nem kötött: minden olyan sorrend megengedett, ami „matematikailag" jó eredményre vezet, azaz az osztás értelmezésével, vagy esetleg a balról jobbra szabállyal nyerhető eredménnyel nem áll ellentétben.*

Ez a szabály lehetővé teszi, hogy a fordítóprogram esetleg optimalizálja a felírt kifejezések kiértékelését, viszont megnehezíti több olyan programozási fogás alkalmazását, amely az ALGOL nyelvben a programozó rendelkezésére áll (ún. mellékhatások). Az egészek osztása is vet fel ide kapcsolódó problémákat, melyekre még visszatérünk.

Megjegyezzük, hogy a precedenciaszabályt lehet abszolút és relatív értelemben felfogni. Ha pl. az aritmetikai kifejezésekben szereplő összes

* Az aritmetikai kifejezések szempontjából kevésbé lényeges az a szemantikai szabály, hogy csak olyan műveletet kell elvégezni, amely befolyásolja a kifejezés értékét. A logikai kifejezésekkel kapcsolatban (lásd a Második fejezet 7. pontját) evvel a szabállyal részletesen foglalkozunk.

szorzást előbb végezzük el, mint a legelső összeadást, akkor abszolút precedenciáról, ha viszont egyetlen operandus két oldalán álló műveleti jelre vonatkozóan igazodunk csak a szabályhoz, akkor relatív precedenciáról beszélhetünk. A FORTRAN nyelv a szabályt csak relatív értelemben tartja kötelezőnek.

Példaként értékeljük ki a (2.6) kifejezést, ha $f=4.0$, $x=10.0$ és $t=3.0$ oly módon, hogy a precedenciaszabályt abszolút értelemben tartjuk be!

1. lépés: Tetszőleges sorrendben elvégezzük az elsődleges műveleteket; azaz a négyzetreemelést és köbreemelést. Eredményül segédmennyiségeket — tényezőket — kapunk:

$$w_1=f^3=64.0 \qquad w_2=x^2=100.0$$

2. lépés: Tetszőleges sorrendben elvégezzük a kijelölt osztásokat és szorzásokat. Mivel egyetlen osztásról és egyetlen szorzásról van szó, csak néhány sorrend lehetséges, pl. az utolsó tagra:

$$w_3 = (w_2/2.0)*t = (100.0/2.0)*3.0 = 50.0*3.0 = 150.0,$$

vagy

$$w_3 = (w_2*t)/2.0 = (100.0*3.0)/2.0 = 300.0/2.0 = 150.0$$

stb.

3. lépés: A tagokat tetszőleges sorredben összevonjuk:

$$w_4 = -w_1+1.0-w_3 = (-64.0+1.0)-150.0 =$$
$$= -63.0-150.0 = -213.0, \text{ vagy}$$
$$w_4 = -w_1-w_3+1.0 = (-64.0-150.0)+1.0 =$$
$$= -214.0+1.0 = -213.0, \text{ vagy}$$
$$w_4 = (-w_3+1.0)-w_1 = (-150.0+1.0)-64.0 =$$
$$= -149.0-64.0 = -213.0 \text{ stb.}$$

Ezek után beszélnünk kell az *aritmetikai kifejezések típusáról.* Minden kifejezés olyan típusú, amilyen aktuális értékének típusa. Az egyes aritmetikai műveletek eredményének típusát a 2. és 3. táblázatban foglaltuk össze. Ezekkel összhangban a kifejezések típusáról a következőket mondhatjuk:

1. *Egész típusú a kifejezés* akkor és csakis akkor, ha benne minden egyes operandus egész típusú. Példaként lásd a (2.4) kifejezést, vagy a következőt:

IG+IH**2−IG/IH

Ha IG=3 és IH=2, akkor a kifejezés aktuális értéke 6.

2. *Valós típusú a kifejezés* akkor és csakis akkor, ha a kitevőktől eltekintve csak valós operandusokat tartalmaz; a kitevők típusa nemcsak valós, hanem egész is lehet. Példák a (2.5) és (2.6) kifejezések.

3. *Kétszeres pontosságú a kifejezés*, ha egy olyan kifejezésben, mely egyébként valós típusú lenne, legalább egy operandus típusa kétszeres pontosságú.

4. *Komplex típusú* az aritmetikai kifejezés, ha benne a kitevőktől eltekintve csak komplex és esetleg valós típusú operandusok fordulnak elő. Például ha CO1 és CO2 komplex típusú operandusok, akkor szintaktikusan helyes a következő komplex típusú kifejezés:

PI*CO1+CO2**3−1.05**T

Térjünk most vissza az egészek osztásának problémájára. Tekintsük a következő egész típusú kifejezést:

I*J/5*K (2.7)

Legyen I=2; J=3; K=16. Milyen értéket vesz fel ez a kifejezés?

Értékeljük ki kifejezésünket a műveleteket különféle — engedélyezett — sorrendben elvégezve, és ügyelve arra, hogy az egész osztás „abszolút értékben lefelé" kerekít! A műveletek elvégzésének sorrendjét jelöljük *zárójelekkel* a következő példák szerint:

((I*J)/5)*K = (6/5)*16 = 16

((I*J)*K)/5 = 96/5 = 19

(I/5)*(J*K) = (2/5)*48 = 0

((I*K)/5)*J = (32/5)*3 = 18

stb.

Láthatjuk, hogy a FORTRAN nyelv szabályai szerint helyesen felírt (2.7) egész típusú vagy *egész kifejezés értéke nincs egyértelműen definiálva!* (Egyes gépi reprezentációk ilyen esetben a balról jobbra szabály szerint értékelik ki a kifejezést.)

Vegyük most a következő valós típusú kifejezést:

$$A*B/C/D \tag{2.8}$$

ahol legyen A = 3E37, B = 4E48, C = 5E52 és D = 6E47. Tegyük fel, hogy olyan számológép értékeli ki kifejezésünket, melyben a legnagyobb ábrázolható valós szám értéke kb. 10^{77}, és a gép olyan, hogy ha egy szorzás ennél nagyobb eredményt adna (ún. lebegőpontos túlcsordulás lépne fel), akkor az eredményt a legnagyobb ábrázolható számmal helyettesíti. Mi ennek a hatása különféle műveleti sorrendek esetén?

$$(A*B)/(C*D) = 10^{77}/10^{77} = 1.0$$

$$((A*B)/C)/D = (10^{77}/5E52)/6E47 = 2E24/6E47 =$$
$$= 0.333E—23$$

$$((A/C)*B)/D = ((0.6E—15)*4E48)/6E47 =$$
$$= 2.4E33/6E47 = 0.4E—14$$

stb. ...

Láthatjuk, hogy a gépi reprezentációban most sem közömbös az egyes műveletek sorrendje, hiszen a túlcsordulások a különféle sorrendek esetén erősen eltérhetnek egymástól. Példánkban az első sorrend két túlcsordulást eredményezett: 10^{85} és 10^{99} helyett számolt a gép 10^{77}-tel. A második sorrendben csak egy túlcsordulás lépett fel (10^{85}), a harmadikban egy sem.

Ha elő akarjuk írni aritmetikai kifejezésekben a műveletek elvégzésének sorrendjét, *zárójeleket* alkalmazunk ugyanúgy, ahogyan az a közönséges matematikai képletekben szokásos. Egyetlen zárójelezéshez egy *bal oldali* és egy *jobb oldali zárójelet* használunk fel. Minden megkezdett zárójelet be kell zárni! A *zárójelpár belsejében* aritmetikai kifejezés áll. A zárójelpárt kívülről úgy kell tekinteni, mintha a benne szereplő aritmetikai kifejezéssel együtt egyetlen operandus lenne. A FORTRAN-ban csakis a kerek zárójelek, „(" és „)" használhatók.

Abból, hogy a zárójelbe zárt kifejezés egyetlen operandust jelöl, következik, hogy a teljes aritmetikai kifejezés kiértékelése a zárójelpárba zárt kifejezés kiértékelésével kezdődik. Többszörösen egymásba skatulyázott zárójelek esetén mindig a „legbelső" zárójel kiértékelése a legelső feladat, akárcsak közönséges matematikai képletek esetén.

Írjunk át most néhány matematikai formulát FORTRAN nyelvbeli aritmetikai kifejezéssé. Vigyázzunk: az osztás művelete mindig az utána következő operandusra vonatkozik és ezért célszerű a törtek nevezőjét zárójelek közé tenni.

A 19. oldalon a (2.1) másodfokú egyenlet megoldóképlete jobb oldalának egy lehetséges átírását adtuk meg a (2.2) aritmetikai kifejezéssel.

A csonkakúp térfogata: $\frac{\pi h}{3}(R^2+Rr+r^2)$. FORTRAN aritmetikai kifejezésben:

```
PI*H*(R*R+R*R1+R1*R1)/3.0
```

Tekintsük a következő kétismeretlenes lineáris egyenletrendszert:

$$ax+by=p$$
$$cx+dy=q$$

ennek a megoldóképlete

$$x=\frac{pd-bq}{ad-bc} \qquad y=\frac{aq-cp}{ad-bc}$$

A jobb oldalak FORTRAN-beli aritmetikai kifejezés formájában, ha a nevező nem nulla:

```
(P*D-B*Q)/(A*D-B*C)
(A*Q-C*P)/(A*D-B*C)
```

Az

$$\frac{\frac{a}{b}-1}{s\left(\frac{s}{d}-1\right)}$$

formulának megfelelő aritmetikai kifejezés

```
(A/B-1.0)/S/(S/D-1.0)
```

Az

$$\frac{\sqrt[n]{a+b}\sqrt[m]{d}}{\sqrt[n+m]{q}}$$

matematikai felírásmódnak a FORTRAN nyelvben az

(A+B)* *(1.0/ZN)*D* *(1.0/ZM)/Q* *(1.0/(ZN+ZM))

kifejezés felel meg. N és M helyett ZN-et és ZM-et használtunk, mivel valós típusú kifejezésünkben valós típusú operandusokat kívántunk szerepeltetni.

Látható, hogy zárójelek alkalmazásával valóban tetszőleges műveleti sorrendet előírhatunk. Maga a matematika a precedenciaszabályt végeredményben kényelmi szempontból használja: segítségével igen sok zárójel kiírásától mentesülünk.

Jelen pontban még egy fogalom bevezetésével kívánjuk bővíteni az aritmetikai kifejezések fogalmát. A FORTRAN nyelvben is megengedett, hogy függvényhivatkozást építsünk be az aritmetikai kifejezésbe, éppen úgy, ahogyan a függvényeket a matematikai képletekben használjuk. Például az

$$a^2+b^2-2ab\cos\gamma$$

formulát a FORTRAN-ban a következőképpen írhatjuk le:

A* *2+B* *2−2.0*A*B*COS (GAMMA)

Láthatjuk, hogy a függvényt valamilyen névvel, ún. *függvényazonosítóval* jelöltük, melyet zárójelek közé zárva követett a független változó. Más példa: az

$$\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{x^2}{2\sigma^2}\right)$$

kifejezés egy lehetséges FORTRAN megfelelője:

1.0/(C*SZ)*EXP(−X* *2/SZ* *2*0.5)

ha C értéke éppen $\sqrt{2\pi}$. Láthatjuk, hogy a függvény argumentumát tartalmazó zárójelpár közé aritmetikai kifejezést is beírha-

4. táblázat

Belső standard függvények

Elnevezés	Definíció	Argumentum száma	Függvény-azonosító	Argumentum típusa*	Függvény típusa*
Abszolút érték	$\|a\|$	1	ABS IABS DABS	R I D	R I D
Csonkítás v. abszolút lefelé kerekítés	$\|a\|$-nál kisebb legnagyobb egész, *a* előjelével ellátva	1	AINT INT IDINT	R R D	R I I
Maradékképzés**	$a_1 (\mathrm{mod}\ a_2)$	2	AMOD MOD	R I	R I
Legnagyobb argumentum kiválasztása	$\mathrm{Max}\,(a_1, \ldots)$	$\geqq 2$	AMAX0 AMAX1 MAX0 MAX1 DMAX1	I R I R D	R R I I D
Legkisebb argumentum kiválasztása	$\mathrm{Min}\,(a_1, \ldots)$	$\geqq 2$	AMIN0 AMIN1 MIN0 MIN1 DMIN1	I R I R D	R R I I D
Konverzió egészből valósba		1	FLOAT	I	R
Konverzió valósból egészbe		1	IFIX	R	I
Előjelezés	$\mathrm{Sign}\,(a_2) \cdot \|a_1\|$	2	SIGN ISIGN DSIGN	R I D	R I D
Pozitív differencia	$a_1 - \mathrm{Min}\,(a_1, a_2)$	2	DIM IDIM	R I	R I
Konverzió duplapontosságúból valósba		1	SNGL	D	R
Valós rész képzés	Re {Z}	1	REAL	C	R
Képzetes rész képzés	Im {Z}	1	AIMAG	C	R
Konverzió valósból duplapontosságúba		1	DBLE	R	D
Komplex szám képzés	$a_1 + a_2\sqrt{-1}$	2	CMPLX	R	C
Komplex konjugált képzés		1	CONJG	C	C

*, ** megjegyzések magyarázata az *5. táblázat* alatt!

5. táblázat

Külső standard függvények

Elnevezés	Definíció	Argumentum száma	Függvény-azonosító	Argumentum típusa*	Függvény típusa*
Exponenciális függvény	e^a	1	EXP DEXP CEXP	R D C	R D C
Természetes logaritmus	$\overset{e}{\log}(a) = \ln(a)$	1	ALOG DLOG CLOG	R D C	R D C
Tízes alapú logaritmus	$\overset{10}{\log}(a)$	1	ALOG1Ø DLOG1Ø	R D	R D
Szinuszfüggvény	$\sin(a)$	1	SIN DSIN CSIN	R D C	R D C
Koszinuszfüggvény	$\cos(a)$	1	COS DCOS CCOS	R D C	R D C
Tangens hiperbolikus	$\tanh(a) = \mathrm{th}(a)$	1	TANH	R	R
Négyzetgyökvonás	$(a)^{1/2}$	1	SQRT DSQRT CSQRT	R D C	R D C
Árkusz tangens főértéke	$\mathrm{arctg}(a)$	1	ATAN DATAN	R D	R D
	$\mathrm{arctg}(a_1/a_2)$	2	ATAN2 DATAN2	R D	R D
Maradékképzés**	$a_1 (\mathrm{mod}\ a_2)$	2	DMOD	D	D
Modulus	Abs {Z}	1	CABS	C	R

* a megfelelő típusokat a típus kezdőbetűjével jelöltük. I = INTEGER, R = REAL, D = DOUBLE PRECISION, C = COMPLEX.

** Definíció: $a_1\ (\mathrm{mod}\ a_2) = a_1 - [a_1/a_2] \cdot a_2$, ahol $[x]$ az a legnagyobb abszolút értékű egész szám, melynek abszolút értéke kisebb, mint x abszolút értéke, előjele pedig megegyezik x előjelével.

tunk. Sőt, mint ahogy a következő példa mutatja, a függvények egymásba skatulyázhatók. Az

$$\ln (x) \operatorname{arc\,tg} (\cos (x+y) \cdot \sin b)$$

formula FORTRAN megfelelője:

ALOG (X) * ATAN (COS (X + Y) * SIN (B))

A függvények jelölésével, azonosítóikkal később foglalkozunk. Használatuknak szabályai aritmetikai kifejezésekben a következők:

1. A függvényazonosító az őt követő zárójelpárba zárt *argumentummal* (többváltozós függvény esetében vesszőkkel elválasztott argumentumokkal) együtt elsődleges aritmetikai kifejezést alkot.

2. Függvényazonosító lehet minden olyan azonosító, melyet vagy maga a FORTRAN nyelv annak definiál, vagy a FORTRAN program annak deklarál. Utóbbi esetben a függvény kiszámítási módját is meg kell adnunk a programban (lásd a Negyedik fejezet 6. pontját és az Ötödik fejezetet)!

Most csupán azokkal a függvényazonosítókkal foglalkozunk, amelyeket a FORTRAN nyelv definiál. Ezek lehetnek ún. *belső* és *külső standard függvények*. A kétféle típus a programozó szempontjából csaknem azonos módon használandó, ezért a köztük levő különbségekkel nem foglalkozunk. Jelöléseiket, jelentésüket és egyéb adataikat a 4. és 5. táblázat adja meg.

Nagyon fontos beszélnünk a függvények és argumentumaik típusáról. Minden függvény meghatározott típusú értéket vesz fel. Ezt a típust nevezzük a függvény típusának, vagy azt mondjuk, hogy ilyen típusú a függvény. Az argumentum (vagy argumentumok) típusának meg kell felelnie a függvény definíciójának. Az argumentum lehet megfelelő típusú aritmetikai kifejezés is.

Mint a 4. és 5. táblázatból látható, a standard függvények azonosítói úgy vannak megválasztva, hogy kezdőbetűik lehetőleg megfeleljenek az automatikus típusdeklaráció szabályának (lásd a Második fejezet 3. pontját), vagy legalábbis kifejezzék a függvény típusát. Ezek közt a standard függvények közt számos konverziós eljárást is találunk, melyek segítségével

eltérő típusú változót vagy kifejezést is beépíthetünk aritmetikai kifejezéseinkbe; például

B ∗ ∗ 2 + C ∗ FLOAT(I ∗ K)

valós típusú kifejezés, melyben az I ∗ K tényező egész típusú, vagy

I + 10 ∗ K − IFIX(Q/T)

egész típusú kifejezés, melyben a Q/T tag valós típusú. Megtalálhatók a standard függvények között a leggyakrabban használt függvények: SIN (X), COS (X), ALOG (X), ALOG10(X), EXP(X), ATAN (X), sőt ezeknek olyan változatai is, melyeket kétszeres pontosságú és komplex típusú kifejezésekben használhatunk. Külön jele van a négyzetgyökvonásnak: SQRT (X).

Ezeknek a függvényazonosítóknak az alkalmazását a FORTRAN nyelv elég bonyolult módon korlátozza. Ezért helyesen járunk el, ha csak a megfelelő standard függvények azonosítására használjuk fel őket.

Néhány példa formulák átírására aritmetikai kifejezéssé, a standard függvények felhasználásával:

A formula: $\sqrt{s(s-a)(s-b)(s-c)}$
a kifejezés: SQRT(S ∗ (S − A) ∗ (S − B) ∗ (S − C))

A formula: $\sqrt{1-\cos^4 x - \ln x^2}$
a kifejezés: SQRT(1.0 − COS (X) ∗ ∗ 4 − ALOG(X ∗ ∗ 2))

A formula: $\cos\,(a^{b^c} - \text{arc tg}\,(b-c))$
a kifejezés: COS (A ∗ ∗ (B ∗ ∗ C) − ATAN(B − C))

6. Relációk

Ha két aritmetikai kifejezés értékét összehasonlítjuk (relációba hozzuk), különféle megállapításokat tehetünk, pl.: egyenlőek, nem egyenlőek, az első nagyobb a másodiknál, az első nagyobb vagy egyenlő a másodikkal stb. Most vegyük magunknak a bátorságot és az értékek ismerete nélkül mondjuk ki, hogy pl. egyenlőek. Két eset lehetséges: ez az állításunk vagy

megfelel a valóságnak, vagy nem. Ilyen állításokat a FORTRAN-ban ún. relációk segítségével írhatunk fel.

A FORTRAN nyelvben hatféle ún. relációjel van definiálva. Ezek jelölése jelentésükkel együtt a következő:

Relációjel	*Jelentése*
.EQ.	egyenlő
.NE.	nem egyenlő
.GT.	nagyobb, mint
.GE.	nagyobb vagy egyenlő
.LE.	kisebb vagy egyenlő
.LT.	kisebb, mint

A pontok a jelölések lényeges részét képezik.

Formálisan egy relációt úgy írunk fel, hogy két aritmetikai kifejezést egy relációjellel kötünk össze. (Ennek speciális esete, amikor valamelyik kifejezés egyetlen változó.) Relációba hozhatók azonos típusú aritmetikai kifejezések, kivéve a COMPLEX típusúakat, továbbá relációba hozhatók egymással a nem azonos típusú aritmetikai kifejezések közül a REAL típusúak a DOUBLE PRECISION típusúakkal. Például az

X**2.GE.4.0 olvasd: $x^2 \geqq 4$ (2.9)

és a

B**2.LT.4.0*A*C olvasd: $b^2 < 4ac$ (2.10)

valamint az

A+B.EQ.C olvasd: $a+b = c$ (2.11)

reláció valós kifejezéseket hasonlít össze az automatikus típusdeklarációs szabály szerint.

Attól függően, hogy X, A, B és C értéke a reláció kiértékelésekor mekkora, maga a reláció „értéke" lehet az igaz, vagy a hamis. Azt mondjuk, hogy a reláció felveszi az igaz vagy a hamis, a FORTRAN szintaxisa szerint a TRUE vagy a FALSE értékeket.

A relációk azok a legegyszerűbb programozási objektumok, amelyek ezeket a logikai értékeket vehetik fel. Kiértékelésük a két aritmetikai kifejezés kiértékelésével kezdődik.

Formális szempontból fel szeretnénk itt hívni arra a figyelmet, hogy valós típusú változók esetén (2.11) mintájára az egyenlőség nagyon kevés esetben teljesül. Sok esetben célszerűbben járunk el, ha (2.11) helyett az

$$\text{ABS(A + B − C).LT.EPSIL} \tag{2.12}$$

relációt írjuk programunkba, ahol EPSIL egy megfelelően kicsire választott szám. Tehát az $|a+b-c|<\varepsilon$ reláció teljesülését követeljük meg az eredeti egyenlőség teljesülése helyett.

7. Logikai változók, műveletek, kifejezések

A programozásban szükség lehet arra, hogy egy reláció logikai értékét, vagy valamely más logikai értéket későbbi felhasználáshoz mint részeredményt megőrizzünk. Ilyen célra logikai változókat alkalmazhatunk.

Az automatikus típusdeklarálás szabálya nem gondoskodik a logikai változók azonosítóiról. Ezeket mindig speciális utasítással kell megadni (lásd a 4.1 pontot). Most célszerűségi okokból a logikai változók jelölésére L betűvel kezdődő azonosítókat fogunk használni.

Ahogyan két számérték közt aritmetikai műveleteket, ugyanúgy két logikai érték közt ún. logikai műveleteket lehet értelmezni. A formális és matematikai logikával összhangban, a FORTRAN nyelv is értelmez ilyen műveleteket, mégpedig a következőket:

1. Logikai szorzás (konjunkció, olvasva „és"), jele: .AND.
2. Logikai összeadás (diszjunkció, olvasva „akár"), jele: .OR.
3. Tagadás (negáció, olvasva: „nem"), jele: .NOT.

Megjegyezzük, hogy ezekből a műveletekből a matematikai logika bonyolultabb műveletei mind felépíthetők.

A kétparaméteres logikai műveleteket 2×2 „bejáratú", ún. értéktáblázatokkal célszerű definiálni, mivel az „argumentumok" csak kétféle logikai értéket vehetnek fel. Az értéktáblázatok a következők:

1. Logikai szorzás: LA.AND.LB

		LB	
		TRUE	FALSE
LA	TRUE	TRUE	FALSE
	FALSE	FALSE	FALSE

(2.13)

A táblázat bemenetein LA és LB értékeit, a megfelelő táblázatelemben az eredményt adtuk meg. Látható, hogy logikai szorzás csak akkor ad TRUE eredményt, ha mindkét komponens értéke TRUE.

2. Logikai összeadás: LA.OR.LB

		LB	
		TRUE	FALSE
LA	TRUE	TRUE	TRUE
	FALSE	TRUE	FALSE

(2.14)

Látható, hogy a logikai összeadás eredménye csak akkor FALSE, ha mindkét komponens értéke FALSE.

3. Tagadás: .NOT.LA

Szemben az előzőekkel, a tagadás egyparaméteres művelet:

	LA	
	TRUE	FALSE
.NOT.LA	FALSE	TRUE

(2.15)

Szemléltessük logikai műveleteinket! Vegyünk két relációt: I.GT.0 és J.GT.0. Ha ezek I és J aktuális értékétől függően teljesülnek, legyen LA, ill. LB értéke TRUE, ellenkező esetben FALSE. Mit jelent akkor a LA.AND.LB kifejezés értéke? Ha LA.AND.LB értéke TRUE, ez azt jelenti, hogy mind I, mind pedig J aktuális értéke nagyobb nullánál; ha LA.AND.LB értéke FALSE, akkor legalább egyikük nem nagyobb nullánál. Viszont LA.OR.LB értéke arra ad választ, hogy I vagy J közül legalább az egyik nagyobb-e, mint nulla; ha igen, akkor LA.OR.LB értéke TRUE, ellenkező esetben FALSE. A .NOT.LA tagadás

értéke pedig arra ad választ, hogy I kisebb, vagy egyenlő-e zérusnál: ha igen, úgy .NOT.LA értéke TRUE, ha nem, úgy FALSE.

Ismerjük tehát a logikai műveleteket. Ezek segítségével három egyszerű logikai kifejezést már fel is írtunk: LA.AND.LB, LA.OR.LB és .NOT.LA. Írjunk most fel néhány bonyolultabbat!

LA.AND.A.LT.B (2.16)

I.EQ.0.OR.J.EQ.2 (2.17)

.NOT.LA.AND.M.EQ.J**2 (2.18)

A.NE.0.0.AND.B**2.GT.4.0*A*C (2.19)

LA.AND.LB.OR.LC (2.20)

LA.AND.(LB.OR.LC) (2.21)

Vizsgáljuk meg a felírt logikai kifejezéseket! Eközben ismerkedjünk meg szerkezetükkel és kiértékelésük szabályaival!

A (2.16) példa szerint logikai művelet végezhető relációval is; ilyenkor először kiértékelhetjük a relációt, majd logikai értékének felhasználásával a logikai műveletet végezzük el. Például, ha a LA logikai változó értéke TRUE, viszont A=3.2 és B=2.5, vagyis az A.LT.B reláció nem teljesül, azaz értéke FALSE, akkor a logikai szorzás (.AND.) művelete a FALSE végeredményre vezet.

A (2.17) logikai kifejezés kiértékelése két reláció kiértékelésével kezdődik: ha I=0, akkor az I.EQ.0 reláció értéke TRUE, ellenkező esetben FALSE; ha J=2, akkor a J.EQ.2 reláció értéke TRUE, ellenkező esetben FALSE. Ha valamelyikük is a TRUE értéket veszi fel, a logikai összeadás művelete már a TRUE végeredményt adja.

A (2.18) kifejezés kiértékelését szintén egy reláció, $m=j^2$ kiértékelésével kell elkezdeni. Ezután bajba jutunk, mert nem tudjuk, hogy az .AND. vagy a .NOT. műveletet értékeljük ki előbb (precedencia?). (2.19) hasonló a (2.17) kifejezéshez, csupán az egyik relációban aritmetikai kifejezések szerepelnek. (2.20) kiértékelése szintén valamiféle predenciaszabály ismeretét tételezi fel. (2.21)-ből látható, hogy a műveletek elvégzésének sorrendjét itt is zárójelekkel lehet szabályozni.

Láthatjuk, hogy logikai kifejezésekbe aritmetikai kifejezések, relációjelek, logikai változók és a logikai műveletek jelei építhetők be zárójelek felhasználásával. A zárójelek alkalmazásának szabályai azonosak az aritmetikai kifejezések kapcsán elmondott szabályokkal. Szükségünk van viszont új precedenciaszabályok felállítására és új szintaktikus osztályok definiálására. Ezek — definícióikkal együtt — a következők:

1. *Elsődleges logikai kifejezések* a következők:

a) *logikai konstans*, azaz a .TRUE. vagy a .FALSE.*
b) *logikai változó azonosító*, pl.: LA
c) *reláció*, pl.: I.LT.J
d) logikai típusú függvény (lásd az Ötödik fejezet 3. pontját).
e) zárójelbe zárt logikai kifejezés, pl.: (LA.OR.LB).

2. *Logikai tényező* lehet vagy egy elsődleges logikai kifejezés, vagy a .NOT. műveleti jel a mögötte álló elsődleges kifejezéssel együtt. Pl.: LA vagy .NOT.LA

3. A *logikai tag* logikai tényezők egy- vagy többelemű sorozata, amelyben az egyes tényezőket az .AND. műveleti jel választja el. Pl.:

LA

LA.AND.LB

.NOT.LA.AND.LB

4. A *logikai kifejezés* logikai tagok egy- vagy többelemű sorozata, amelyben az egyes tagokat az .OR. műveleti jel választja el. Pl.:

LA

LA.OR.LB.OR.LC

LA.AND.LB.OR.LC

Az utolsó kifejezésben például két tag szerepel: LA.AND.LB, illetve LC, melyek közül az első két tényezőből, LA-ból és LB-ből tevődik össze.

* Ahogyan a számértéket a programban számkonstanssal is megadhatjuk, ugyanúgy a logikai érték megadásához a logikai konstansokat is felhasználhatjuk. A pontok a konstansokhoz hozzátartoznak.

A definíciókból következik a logikai kifejezésekben szereplő műveleti jelek precedenciája:

1. elsődlegesek az aritmetikai műveletek jelei, az egymásközti precedencia betartásával.
2. másodlagosak a relációjelek
3. harmadlagos a .NOT. műveleti jel
4. negyedleges az .AND. műveleti jel
5. ötödleges az .OR. műveleti jel.

Most már érthető (2.17) korábbi kiértékelési módja; (2.18) kiértékelése esetén előbb a .NOT. műveletet és később az .AND. műveletet kell végrehajtani; (2.20)-ban előbb az .AND. műveletet, később az .OR. műveletet kell végrehajtani.

Látható, hogy a logikai kifejezésekbe aritmetikai kifejezéseket lehet beépíteni. Ez azt sugallja, hogy ne különböztessük meg az aritmetikai és a logikai kifejezéseket, hanem általában kifejezésről beszéljünk. Ebben az esetben természetesen az operandusok típusának meg kell felelniök a rájuk ható műveletek kívánalmainak. Ez a tendencia az ALGOL nyelvben méginkább érezhető.

A logikai műveletek kiértékelésének szabályai hasonlóak az aritmetikai kifejezések kiértékelésének szabályaihoz:

1. Különféle precedenciájú műveleteket — ha a sorrend befolyásolja az eredményt — precedenciájuk sorrendjében kell kiértékelni.
2. Amennyiben az 1. szabály nem korlátoz, a műveletek kiértékelésének sorrendje tetszőleges.

Tekintettel arra, hogy a most következő szabálynak jelen esetben nagyobb jelentősége van, mint az aritmetikai kifejezéseknél (lásd a 28. oldali lábjegyzetet), ténylegesen ki is mondjuk, hogy:

3. Egy műveletet csak akkor kell feltétlenül végrehajtani, ha a kifejezés értékét befolyásolja a művelet végrehajtása is.

Hogy jobban lássuk szabályaink jelentőségét, példaképpen értékeljük ki a

LA.OR.LB.OR.LC.AND.(LP.OR.LQ).AND.(I ** 2.LT.9.OR.LD) (2.22)

logikai kifejezést, ha

LA = LC = LP = LD = FALSE, LB = LQ = TRUE és I = 2

E_i-vel részeredményeket jelölünk!

Két úton járjunk most el:

1. A precedenciákhoz szigorúbban alkalmazkodó módszer szerint:

E1 = LP.OR.LQ = TRUE

E2 = I * * 2 = 4

E3 = 4.LT.9 = TRUE

E4 = E3.OR.LD = TRUE

E5 = LC.AND.E1 = FALSE

E6 = E5.AND.E4 = FALSE

E7 = LA.OR.LB = TRUE

E8 = E7.OR.E6 = TRUE, ami a végeredmény.

2. Az ALGOL nyelvben használt balról jobbra szabályhoz szigorúan alkalmazkodó módszer szerint (így is szabad!):

E1 = LA.OR.LB = TRUE, ami már a végeredmény, és ezért tovább nem is kell számolni, hiszen akárhány .OR. tagolná is tovább kifejezésünket, a közrezárt logikai tagok értékétől a végeredmény már nem függ: volt már egy TRUE értékű tag. Az eredmény természetesen nem függ a kiértékelési módszertől.

Más példa erre az utóbbi módszerre:

Értékeljük ki a fenti értékekkel a

LA. OR. I * * 2. LT. 1. AND. (LP. OR. LQ). OR. LC. OR. {folytatás...} (2.23)

logikai kifejezést!

E_1 = LA = FALSE

tehát a következő tagot még ki kell értékelni.

E_2 = I * * 2 = 4

E_3 = 4.LT.1 = FALSE

E_4 = E_3.AND. {egy tényező} = FALSE. Mivel az első tényező FALSE volt, nem kell kiértékelni a második tényezőt. Át kell térni tehát a harmadik tagra

E_5 = LC = TRUE = E_6, ez a végeredmény, függetlenül attól, mi van a harmadik .OR. jobb oldalán.

8. Gyakorlatok

Ezekben a gyakorlatokban az azonosítók kezdőbetűjét a szokásos módon típusazonosításra használjuk. I, J, K, M, N egész típusú, az L kivételesen logikai típusú változót jelöl.

1. Írjuk át aritmetikai kifejezéssé a következő matematikai formulákat:

a) $$\sqrt{\frac{1-\cos(\alpha+\beta)}{1+\cos(\alpha+\beta)}}\cdot\sqrt{\frac{1-\cos(\alpha-\beta)}{1+\cos(\alpha-\beta)}}$$

b) $$\frac{1}{(1+\alpha)^n}\cdot\frac{(1+\alpha)^n-1}{\alpha}$$

c) $$\frac{\exp\left\{-\dfrac{(x-a)^2}{2\sigma^2}\right\}}{\sqrt{2\pi\sigma^2}}$$

d) $$\sqrt{\frac{\varkappa-1}{\varkappa}\cdot\frac{4\alpha^2 gh}{1+4h^2(\alpha^2+\beta^2)}}$$

e) $$A\cdot e^{\frac{z}{2H}}\cdot\sin\alpha x\cdot\cos\beta z\cdot\cos\omega t$$

2. A következő aritmetikai kifejezések szintaktikusan hibásak. Mi a hiba? Mi lehet a helyes felírás?

a) (A/B)/–C+(A/B)*C**2**3

b) SIN X**2+SIN**2X

c) A.E–2**B

d) –B±SQRT(B**2–4*A*C)/2*A

e) C**A/2–I

3. Írjuk vissza szabványos matematikai alakba a következő aritmetikai kifejezéseket:

a) A/B/B1*A1–SQRT(1.0+COS(2.0*X))

b) (–Q+SQRT(Q**2+P**3))**0.33333333

4. Írjunk (standard könyvtári függvények alkalmazása nélkül) aritmetikai kifejezést, melynek értéke 0, ha I páros, és 1, ha I páratlan pozitív egész.

5. Írjunk aritmetikai kifejezéseket, melyek az I egész változó különböző pozitív értékeire a következő értékeket veszik fel:

a) $$f_1(\mathsf{I}) = \begin{cases} \mathsf{I}, & \text{ha} \quad \mathsf{I} \text{ páros} \\ \mathsf{I}+1, & \text{ha} \quad \mathsf{I} \text{ páratlan} \end{cases}$$

b) $$f_2(\mathsf{I}) = \begin{cases} \mathsf{I}, & \text{ha} \quad \mathsf{I} \text{ páratlan} \\ \mathsf{I}+1, & \text{ha} \quad \mathsf{I} \text{ páros} \end{cases}$$

c) $$f_3(\mathsf{I}) = \begin{cases} \mathsf{I}, & \text{ha} \quad \mathsf{I} \text{ páratlan} \\ \mathsf{I}-1, & \text{ha} \quad \mathsf{I} \text{ páros} \end{cases}$$

6. Írjunk (standard könyvtári függvények alkalmazása nélkül) aritmetikai kifejezést, mely I és M pozitív egész értékeire az osztási maradékot adja eredményül! (A **4.** feladat ennek speciális esete $\mathsf{M}=2$-re!)

7. Legyen I1, I2 és M pozitív egész úgy, hogy $1 \leqq \mathsf{I1} \leqq \mathsf{M}$ és $1 \leqq \mathsf{I2} \leqq \mathsf{M}$. Írjunk aritmetikai kifejezéseket, melyek értékei:

a) $$f_1 = \begin{cases} \mathsf{I1}+\mathsf{I2} & \text{ha} \quad \mathsf{I1}+\mathsf{I2} \leqq \mathsf{M} \\ \mathsf{I1}+\mathsf{I2}-\mathsf{M} & \text{ha} \quad \mathsf{I1}+\mathsf{I2} > \mathsf{M} \end{cases}$$

b) $$f_2 = \begin{cases} \mathsf{I1}-\mathsf{I2} & \text{ha} \quad \mathsf{I1}-\mathsf{I2} \geqq 1 \\ \mathsf{I1}-\mathsf{I2}+\mathsf{M} & \text{ha} \quad \mathsf{I1}-\mathsf{I2} < 1 \end{cases}$$

Következő feladataink megoldásához használjuk fel a belső standard függvényeket!

8. Oldjuk meg a **4.** és **5.** feladatot, feltéve, hogy I negatív értéket is felvehet!

9. Az $a_n = a_1+(n-1)d$ összefüggés megadja a számtani sor n-edik tagját. Legyen $d>0$ és $b>a_1>0$. Válasszuk ki a számtani sor azon legnagyobb tagját, mely még kisebb, mint b. Írjunk fel olyan aritmetikai kifejezést, melynek értéke éppen ennyi!

10. Legyen A és B értékére igaz, hogy $0 \leqq \mathsf{A} \leqq \mathsf{B}$. Írjunk fel olyan aritmetikai kifejezést, melyre:

a) $$f_1(\mathsf{A}) = \begin{cases} \mathsf{A} & \text{ha} \quad \mathsf{A} \leqq \mathsf{B}/2 \\ \mathsf{B}-\mathsf{A} & \text{ha} \quad \mathsf{A} \geqq \mathsf{B}/2 \end{cases}$$

b) $$f_2(\mathsf{A}) = \begin{cases} 1 & \text{ha} \quad \mathsf{A} = \mathsf{B}, \text{ vagy } \mathsf{A} = 0 \\ 0 & \text{egyébként} \end{cases}$$

Hogyan viselkedik a feladatok megoldása, ha $\mathsf{A}>\mathsf{B}$, vagy $\mathsf{A}<0$?

11. Legyen B tetszőleges valós szám. Adjunk meg olyan aritmetikai kifejezést, melynek B1 értékére igaz, hogy $-\pi/2 \leqq \mathsf{B1} \leqq +\pi/2$ és melyre $\sin \mathsf{B1} = \sin \mathsf{B}$!

12. Számítsuk ki a következő logikai kifejezések értékét, ha LA és LB logikai értéke TRUE, LC és LD logikai értéke FALSE, valamint X=3.5; Y=4.2; A=5.1; B=6.1; C=7.1; D=8.1; I=3 és J=2:

a) LC.OR.LA.AND.I.EQ.J.OR.LD

b) X.LT.A.AND.X.GT.FLOAT(I).OR.LD

c) B.GT.A.AND.B.LT.C.AND.C.LT.D.OR.LC.AND.LD

d) X* *2+Y* *2.LT.25.AND.X.GT.Y.OR.J.EQ.I

e) (LC.OR.LA).AND..NOT.LD.OR.J.GT.I

13. Írjunk fel olyan logikai kifejezést, melynek aktuális értéke TRUE, ha φ az első síknegyedbe mutató szögérték, különben pedig FALSE! Tegyük fel, hogy $|\varphi| \leqq 2\pi$!

14. Legyen adva a pozitív, 1-nél nagyobb A valós szám. Írjunk fel olyan aritmetikai kifejezést, melynek értéke kettőnek az a lehető legnagyobb egész kitevős hatványa (2* *N), mely A-nál még kisebb!

Harmadik fejezet

VÉGREHAJTHATÓ UTASÍTÁSOK

1. Az utasítások típusai a FORTRAN nyelvben

A FORTRAN utasításokat két csoportba soroljuk. A *végrehajtható* utasításokat általában a célprogram hajtja végre a futtatás során. A *nem végrehajtható* (vagy *deklaratív*) utasítások főként a fordítóprogramnak szólnak. Az előbbiek részletes ismertetése főként ennek a fejezetnek, az utóbbiaké pedig a Negyedik fejezetnek a tárgya.

A végrehajtható utasítások közé soroljuk az értékadó utasításokat, az ún. vezérlőutasításokat és a perifériakezelés több utasítását. Néhány vezérlőutasítás (CALL, RETURN) tárgyalása az Ötödik fejezetbe, a perifériakezelés egésze pedig a Hatodik fejezetbe került.

A végrehajtható utasítások ismeretében már el lehet készíteni olyan programrészleteket, amelyek gyakorlatban előforduló problémákat oldanak meg. Ezért — néhány illusztratív példától eltekintve — a gyakorlatanyagot a fejezet végére vontuk össze.

2. Értékadó utasítások

Az értékadó utasítás három részből áll: *bal oldali változóból*, az *értékadás jeléből* és egy *jobb oldali kifejezésből*. Az értékadás jele a közismert egyenlőségjel: „=". Formálisan:

$$\mathrm{V} = e \qquad (3.1)$$

ahol V egy változó azonosító, = az értékadás jele és e egy kifejezés, mely lehet különféle típusú aritmetikai kifejezés vagy logikai kifejezés is.

Az értékadó utasítás hatására a V változó elveszti korábbi értékét (ha volt), és felveszi az e kifejezés aktuális értékét. Az értékadás csak abban az esetben hajtható végre egyszerűen,

ha a V változó típusa megfelel az *e* kifejezés típusának. Aritmetikai változó nem vehet fel logikai értéket és logikai változó nem vehet fel számértéket. Ezért kétféle értékadó utasításról szokás beszélni: logikai és aritmetikai értékadó utasításról.

a) Logikai értékadó utasítás

Ha a (3.1) képletben V és *e* típusa egyaránt logikai, akkor logikai értékadó utasításról beszélünk. Példák:

```
LA=LB.AND.LC                (3.2)
LB=J.LT.12.OR.I.GT.0        (3.3)
LC=.TRUE.                   (3.4)
```

Megjegyezzük, hogy a gyakorlati programozásban csak a (3.4) alakú értékadó utasításban szokott ún. *logikai konstans* előfordulni. A pontok a logikai konstansok lényeges részei: a konstansok .TRUE. és .FALSE., értékük rendre: TRUE és FALSE!

b) Aritmetikai értékadó utasítás

Ha a (3.1) képletben V és *e* egyaránt aritmetikai típusú, akkor aritmetikai értékadó utasításról beszélünk. Ha V és *e* mindketten aritmetikai típusúak ugyan, de típusuk nem megegyező, akkor a 6. táblázat szerint a bal oldali változó *e* megfelelő függvényének (lásd 4. táblázat) értékét veszi fel; néhány esetben pedig meg van tiltva a típuspárosítás használata.

A következő két utasításban V és *e* típusa egyező: az elsőben INTEGER, a másodikban REAL:

```
I = 2*J+K**2                          (3.5)
F = (B**2-4.0*A*C)/4.0/A**2           (3.6)
```

Egy példa arra az esetre, mikor V REAL, *e* pedig INTEGER típusú:

```
F = 2*J+K+1
```

És a „fordított" eset:

```
I=SQRT(A)                             (3.7)
```

6. táblázat

Típuseltérések aritmetikai értékadásnál

		e típusa			
		INTEGER	REAL	DOUBLE PRECISION	COMPLEX
V típusa	INTEGER	E	IFIX	SNGL és IFIX	TILOS
	REAL	FLOAT	E	SNGL	TILOS
	DOUBLE PRECISION	FLOAT és DBLE	DBLE	E	TILOS
	COMPLEX	TILOS	TILOS	TILOS	E

Az E betű közönséges értékadást jelöl. Megjegyezzük, hogy az SNGL és IFIX konverziók „abszolút értékben lefelé" kerekítenek. A kétszeres pontosságú és egész típus között a konverzió két lépésben, a valós típuson keresztül megy végbe.

Az utolsó példa kapcsán fel kell hívni az olvasó figyelmét egy gyakori programozási hibára:

Legyen az *A* valós típusú változó „szándékolt" értéke 4.0. Elképzelhető, hogy kerekítési hibák folytán a gép 4-nél kb. 10^{-12}-nel kisebb értékkel számol. Ha a SQRT négyzetgyökvonó eljárás is lefelé hibázik „egy kicsit", SQRT(A) értéke biztosan kisebb, mint 2. Értékadás során a 6. táblázat szerint REAL típusú *e* és INTEGER típusú V esetében az IFIX konverziós eljárás kerül alkalmazásra és így a (3.7) utasítás I-nek nem a 2, hanem az 1 értéket adja! Ezért helyesebb (3.7) helyett az

$$\text{I} = \text{SQRT(A)}+1\text{E}-8 \qquad (3.8)$$

utasítást használni. A 10^{-8} érték megfelelő biztonságot nyújt a fent leírt effektus ellen.

Sok FORTRAN gépi reprezentáció ismeri az ún. *többszörös értékadó utasítást* is. Kétféle felírási módja szokásos:

$$V_1 = V_2 = \ldots = V_n = e \qquad (3.9)$$

vagy

$$V_1, V_2, \ldots, V_n = e \qquad (3.10)$$

Az utasítás végrehajtásakor *e* egyszer értékelődik ki, majd értékét rendre felveszik a V_n, V_{n-1}, ..., V_1 változók. Ilyenkor megkövetelik, hogy a V-k típusa megegyezzék egymással.

Gyakran célszerű, hogy egyetlen bonyolultabb értékadó utasítást több utasításra bontsunk szét. Legyen például a feladat az

$$y = f(a, b) = \frac{a^2+b^2+\sin(a^2+b^2)+\ln\sin(a^2+b^2)}{a^2+\sqrt{b^2+\sin(a^2+b^2)}+\ln\sin(a^2+b^2)} \tag{3.11}$$

függvényérték kiszámítása *a* és *b* ismeretében. Megoldható a feladat egyetlen értékadó utasítással is:

```
      Y = (A**2+B**2+SIN (A**2+B**2)
     1 +ALOG(SIN (A**2+B**2)))/(A**2+SQRT(
     2 B**2+SIN (A**2+B**2))+ALOG(SIN (A**
     3 *2+B**2)))                                (3.12)
```

Ugyanezt a feladatot oldja meg a következő programrészlet:

```
      W1 = A**2
      W2 = B**2
      Y  = SIN (W1+W2)                           (3.13)
      W1 = W1+ALOG(Y)
      W2 = W2+Y
      Y  = (W1+W2)/(W1+SQRT(W2))
```

Ebben a programrészletben felhasználtuk a W1 és W2 segédváltozókat (munkaváltozókat), sőt magát Y-t is különféle részeredmények tárolására. (3.13) végrehajtása során a számológép csak kétszer emel négyzetre és egyszer keres szinuszt, ill. logaritmust. Ugyanez nem állítható a (3.12)-ről.

A formulakiértékelés modern elmélete megteremtette annak a lehetőségét, hogy olyan fordítóprogramot írhassunk, amelyik például a (3.12) kifejezést egy a (3.13) programrészlethez hasonló minta szerint fordítja le. Korántsem bizonyos azonban, hogy egy bizonyos számológép FORTRAN fordítóprogramja ilyen; lehet, hogy elvégez bizonyos optimalizáló tevé-

kenységet, de az is lehet, hogy csak keveset, vagy semmit sem. Így, ha ehhez fontos érdek fűződik, bontsuk le mi magunk bonyolultabb kifejezéseinket hasonló programrészletekre. Megjegyezzük, hogy például ALGOL programok esetén a szigorú balról jobbra szabály miatt csak ez az utóbbi út járható.

3. Vezérlő utasítások és címkék. ASSIGN utasítás

Mint az előző (3.13) programrészlet is mutatja, a FORTRAN utasítások felírásuk sorrendjében kerülnek végrehajtásra, hacsak evvel ellenkező végrehajtási módról külön intézkedés nem történik. A *vezérlő utasítások* szerepe éppen az, hogy a felírási sorrendtől eltérő megfelelő végrehajtási sorrendeket írjanak elő. Ezt általában úgy érik el, hogy megnevezik a soron következő utasítást, melynek ehhez természetesen „nevet kell adni". Ezek a „nevek" a FORTRAN nyelvben a *címkék*.

Ha egy FORTRAN utasítást címkével akarunk ellátni, akkor az utasítás első sorának 1—5 karakterpozíciójára 5-nél nem több jegyű egész számot írunk. Ez a számjegyekből álló karaktersorozat az utasítás címkéje. A vezérlésátadó utasításokba ezt a karaktersorozatot kell beírni, ha erre a címkére akarunk ugratni, azaz ha a program végrehajtását a címkével megjelölt utasítással akarjuk folytatni.

Speciális esetben a címkét a vezérlésátadó utasításban egy INTEGER típusú változó azonosító helyettesítheti (l. *kijelölt* GO TO *utasítás* a következő pontban). Ekkor e változó értéke címke kell, hogy legyen. Az ASSIGN utasítás arra szolgál, hogy INTEGER típusú változónak egy konkrét címkét adjon értékül. Alakja:

$$\text{ASSIGN } l \text{ TO } i \qquad (3.14)$$

ahol l cimke, i pedig INTEGER változó azonosító. Például:

$$\text{ASSIGN 118 TO KAPCS} \qquad (3.15)$$

$$\text{ASSIGN 21 TO IZIDOR} \qquad (3.16)$$

Az ilyenfajta utasítások későbbi felhasználás céljára „elraktározzák" a megfelelő címkéket. Hatásuk nem azonos a

$$\text{KAPCS} = 118 \qquad (3.17)$$

$$\text{IZIDOR} = 21 \qquad (3.18)$$

utasításokkal és egymást nem is helyettesíthetik, hiszen a 118 karaktersorozat nem azonos a 118 egész típusú számmal.

Magukat a vezérlő utasításokat jelölésük alapján csoportosítjuk. Beszélni fogunk különféle GO TO utasításokról, aritmetikai és logikai IF utasításról, ciklus, vagy DO utasításról, CONTINUE utasításról, végül a megállító utasításokról, azaz a STOP és a PAUSE utasításról.

a) Feltétlen vezérlésátadás. A feltétlen és a kijelölt GO TO utasítás

Ha azt akarjuk előírni, hogy egy utasítás után ne a sorrendben reá következő kerüljön végrehajtásra, hanem egy meghatározott másik, ezt az ún. feltétlen GO TO utasítással érhetjük el. Ennek formája:

GO TO *l* (3.19)

ahol *l* címke.

Tekintsük például a következő programrészletet:

```
   |  |F = A+B
   |  |GO TO 13        (3.20)
88 |  |G = A-B
13 |  |C = F+G
```

A részlet második utasítása után, a GO TO 13 utasítás hatására a 13-as címkéjű utasítás hajtódik végre. Ilyen esetben azt mondjuk, hogy a „vezérlés” a 13-as címkére adódik át, vagy ugrás történik a 13-as címkére stb.

Magának a GO TO utasításnak is lehet címkéje. Ha véletlenül ugyanazt a címkét viseli, mint amelyre ugrat, akkor ha egyszer erre az utasításra sor került, a program ezt az utasítást fogja ismételni a végtelenségig. Például:

```
125 |  |GO TO 125
```

Úgy is mondhatjuk, hogy a program *„végtelen ciklusba”* került. A fenti ciklus például egyetlen utasításból áll. Programozói hiba sokszor vezethet több utasítást tartalmazó végtelen ciklushoz, amelyekből hasonlóképpen nincs „kijárat”.

Ha a GO TO utasításban a címke helyére egész változót írunk, melyet zárójelbe zárva és vesszővel elválasztva címkelista

követ, például

GO TO KAPCS, (10, 11) (3.21)

GO TO IZIDOR, (3, 8, 10, 15) (3.22)

akkor ún. *kijelölt* GO TO *utasítást* írtunk fel. Az ugrás ilyenkor az egész változó értékének megfelelően történik. Ez az érték nem lehet tetszőleges, hanem szükséges, hogy megegyezzék az utána álló címkelista valamelyik értékével. Az egész változó ezt az értéket egy előzőleg végrehajtott ASSIGN utasításban kapta. Ha az egész változó több ASSIGN utasításban kapott értéket, mindig az utoljára végrehajtott ASSIGN utasítás a mérvadó.

Az ASSIGN utasítások és a megfelelő kijelölt GO TO utasítások közt tehát szoros kapcsolat van. Lássunk erre egy példát*:

```
       :
      ASSIGN 10 TO KAPCS
      R=15.0
      GO TO 153
10    BERUH1 = 48.6*F+125.4*T
      R=23.4
      ASSIGN 11 TO KAPCS
      GO TO 153
11    BERUH2 = 81.3*F+42.8*T
       :                                    (3.23)
       :
153   W = R**2
      F = 12.56637*W
      T = 4.18879*R*W
      GO TO KAPCS, (5,10,11,13,47)
       :
```

* Itt és a továbbiakban a példa szempontjából nem lényeges utasításokat gyakran elhagyjuk, és helyüket függőleges vagy vízszintes pontozással, esetleg a „folytatás" szóval jelöljük.

A KAPCS változó először értékként felveszi a 10-es címkét, R pedig a 15.0 értéket, majd a részlet végére, a 153-as címkére történik ugratás. Itt az R sugarú gömb F felületét és T térfogatát számítjuk ki, és a KAPCS változó jelentésének megfelelően visszaugrunk a 10-es címkére. Itt a felszínt és térfogatot valamilyen empirikus formula szerint beruházási költség számítására használjuk fel. Ezután az egész procedurát más állandókkal megismételjük. Látható, hogy a 153-as címkétől kezdődő programrészletet többször használtuk fel. Hasonló hatást érhetünk el a *d)*-ben tárgyalandó kiszámított GO TO utasítással (lásd a 4. példát a 62. oldalon) is.

Egyes gépi reprezentációk megengedik, hogy a kijelölt GO TO utasításból a címkelista elmaradjon. Ilyen engedmény nélkül nem is célszerű az utasítás használata, hiszen az ASSIGN utasítás elég sok kényelmetlenséget okoz. Helyette a kiszámított GO TO utasítást használják. Viszont a lista elhagyása lehetővé teszi, hogy egy programrészletet ott és annyiszor használjunk fel, ahány helyen csak jól esik. Így mód van valamiféle eljáráshívás szimulálására, ami hasonlít a CALL utasítás (l. az Ötödik fejezet 3. pontját) használatára, vagy az ALGOL nyelv eljáráshívására.

b) Elágaztatás előjel és nulla szerint. Az aritmetikai IF utasítás

A FORTRAN nyelv igen célszerűen felhasználható és a számológépben igen gyorsan végrehajtható jellegzetes utasítása az *aritmetikai* IF *utasítás*. Alakja:

$$\mathrm{IF}(e)\ l_1,\ l_2,\ l_3 \tag{3.24}$$

ahol e egész, valós, vagy kétszeres pontosságú aritmetikai kifejezést, l_1, l_2 és l_3 pedig vesszőkkel elválasztott címkéket jelöl. (Lehetnek e címkék között azonosak is!) Például:

IF(I−J)10,11,7 (3.25)

IF(K)13,13,14 (3.26)

IF(B∗∗2−4.0∗A∗C)15,16,16 (3.27)

IF(ABS(COS(X))−EPS)20,17,20 (3.28)

Az utasítás hatására kiértékelésre kerül a zárójelben levő e aritmetikai kifejezés. Ha ennek értéke negatív, ugrás következik az első (l_1) címkére, ha nulla, akkor a középső (l_2) címkére, ha pozitív, akkor pedig az utolsó (l_3) címkére.

A (3.25) példában az aritmetikai kifejezés két egész szám különbsége. Tehát az ugrás előtt azt vizsgáljuk, hogy egyenlőek-e, illetve melyikük nagyobb. A (3.26) példa szerint az *e* kifejezés lehet egyetlen változó azonosító. Ebben az esetben a vizsgálat az illető változó értékére vonatkozik. Be lehet építeni hosszabb valós kifejezést, pl. a másodfokú egyenlet diszkriminánsát is az aritmetikai IF utasítás zárójelébe [l. (3.27) példa]. Ez azonban általában nem célszerű, mert a kifejezés számértéke elvész, és ha a továbbiakban szükség lenne reá, újra ki kell számítani. A (3.28) példa szerint az aritmetikai kifejezésben előfordulhatnak különféle függvények is. Ha azt akarjuk megállapitani egy valós, vagy kétszeres pontosságú értékről, hogy nulla-e, célszerűbb, ha azt vizsgáljuk, hogy abszolút értéke kisebb-e egy megadott kis pozitív ε-nál. Ugyanis a számológépek számábrázolási pontossága olyan nagy, hogy sok esetben teljesen felesleges ilyen szigorúsággal megkövetelni egy mennyiségtől, hogy az nulla legyen. Különben is, ha ez a mennyiség számítások eredményeként adódik, igen nagy kerekítési hiba terhelheti. Ezért az egzakt nullát a numerikus gyakorlatban annak megfelelő környezete szokta helyettesíteni.

Alkalmazási példaként írjunk egy olyan FORTRAN programrészletet, amely az $ax^2+bx+c = 0$ másodfokú egyenlet valós gyökeit számítja ki. Tegyük fel, hogy $a \neq 0$.

```
      W = 2.0*A
      D = B**2-4.0*A*C
      IF(D)20,18,16
16    D=SQRT(D)
      X1 = (D-B)/W                         (3.29)
      X2 = (-D-B)/W
      GO TO 22
18    X1=(-B)/W
      X2=X1
      GO TO 22
      ⋮
```

Programrészletünk pl. komplex gyökök esetén a 20-as címkére adja a vezérlést.

Az ICT FORTRAN megengedi, hogy az aritmetikai IF utasítás címkéi közt a nulla számjegy is előforduljon, mint címke. Az ilyen „szimbolikus címke" hatására a vezérlés a rákövetkező FORTRAN utasításra adódik át. Ezek szerint a (3.29) programrészlet 3. és 4. utasítását a következő módon is fel lehetett volna írni:

```
      IF(D)20,18,0
      D = SQRT(D)
```

c) Feltételes utasítás. A logikai IF utasítás

Egy FORTRAN utasítást egy feltétel teljesülésétől függően kihagyathatunk vagy elvégeztethetünk a *logikai* IF *utasítás* felhasználásával, melynek formája:

IF(LA)*u* (3.30)

ahol LA tetszőleges logikai kifejezés (pl.: reláció, logikai változó stb.), *u* pedig tetszőleges olyan FORTRAN utasítás, amely nem logikai IF utasítás, nem ciklusutasítás (l. 5. pontot), és melynek nincsen címkéje. Például:

```
      IF(A.GT.B) GO TO 125
      IF(A.LT.0.0)A = -A                (3.31)
 113  IF(I.GT.0.AND.J.LT.5) GO TO 8
      ⋮
```

Az utasítás végrehajtása a logikai kifejezés értékelésével kezdődik. Amennyiben annak értéke TRUE, akkor *u* végrehajtása következik. Ha értéke FALSE, *u* végrehajtása elmarad és a gép áttér a következő FORTRAN utasítás végrehajtására.

Tekintsük a fenti — 3 db logikai IF utasításból álló — programrészletet! Ha $a > b$, akkor elugrik a program a 125-ös címkére. Ha $a \leq b$, sorrakerül a második utasítás. Ha $a < 0$, akkor a előjelet vált. A harmadik utasítás ugrást ír elő a 8-as címkére, ha egyidejűleg fennáll $i > 0$ és $j < 5$. Itt látható az is, hogy magának a logikai IF utasításnak lehet címkéje, csupán a zárójel után következő *u* utasításnak nem!

Több gépi reprezentációban a célprogram a logikai műveleteket viszonylag lassan hajtja végre. Ezért ajánljuk, hogy amennyiben elkerülhető, ne használjuk a logikai IF utasítást. A (3.31) programrészlet helyett pl. a következő írható:

```
      IF(A-B)15,15,125
15    A=ABS(A)
113   IF(I)17,17,16                    (3.32)
16    IF(J-5)8,17,17
17    :
```

Ezzel — a hivatkozási FORTRAN-ban — csupán az a kényelmetlenség jár, hogy a soronkövetkező utasítást fel kell címkéznünk, amire egyébként nem volna szükség.

d) Többirányú elágaztatás. A kiszámított GO TO utasítás

Programok többirányú elágaztatására a FORTRAN nyelv még egy — a kijelölt GO TO utasításhoz hasonló — lehetőséget biztosít: az ún. *kiszámított* GO TO *utasítást*. Az utasítás alakja:

$$\text{GO TO } (l_1, l_2, \ldots, l_n), i \tag{3.33}$$

ahol az $l_1, l_2, \ldots, l_n$ sorozat címkék vesszőkkel elválasztott n elemű sorozata, i pedig INTEGER változó, melynek aktuális értékére a (3.33) utasítás végrehajtásakor fenn kell állnia az

$$1 \leqq i \leqq n \tag{3.34}$$

összefüggésnek. Az utasítás hatása azonos a

$$\text{GO TO } l_i \tag{3.35}$$

utasításéval, ahol l_i a fenti címkelista i-edik eleme. Tehát i megmondja, hogy a lista hányadik elemének megfelelő utasításra kell ugrani.

A lista elemei között akárhány egyenlő is lehet.

Az ICT FORTRAN megengedi a listában a 0 „szimbolikus címke" használatát ugyanúgy, mint az aritmetikai IF utasításban. Ennek hatására most is a reákövetkező FORTRAN utasításra adódik át a vezérlés.

e) Példák

Az eddig megismert utasítások felhasználásával elvileg minden olyan számítás beprogramozható, melyet a FORTRAN nyelvben be lehet programozni. Igaz, hogy nem mindig kényelmesen. Általában további utasításokra van még szükség az eredmények kinyomtatására, vagy — ha adataink változnak — az adatok beolvasására, valamint a program megállítására. De ezen a helyen mód nyílik arra, hogy eddigi utasításaink használatát példákon szemléltessük és bemutassunk néhány olyan fogalmat (ciklus, ciklusváltozó stb.), melyek a gyakorlati programozásban rendkívül hasznosak.

A példák megoldásaként programrészleteket írtunk fel. Feltételezzük mindig, hogy ezek a részletek működőképes program belsejéből valók, tehát a részletekben szereplő változóknak van értékük, a részletet megelőzi a tényleges programban minden olyan utasítás, mely a működéshez szükséges (deklarációk stb., lásd a Negyedik és Ötödik fejezetet), és a részletet is követik megfelelő utasítások, például eredménykiíró vagy megállító utasítások. Sokszor ezekben a csatlakozó részekben elhelyezett címkére hivatkozik a részletek néhány utasítása.

Első két példánkban a *programciklus* fogalmát mutatjuk be:

1. Írjunk olyan programrészletet, amely addig von le a pozitív FI-ből 2π-t, míg értéke 2π alá nem csökken! Tegyük fel, hogy a 2π számértéket az AKETPI változó tartalmazza! Egy megoldás:

```
16   | IF(FI-AKETPI) 18,17,17
17   | FI = FI-AKETPI                        (3.36)
     | GO TO 16
18   | ...
```

A programrészlet „működése” szemléletes: a 16-os címkéjű utasítás megvizsgálja, hogy egy feltétel — mégpedig FI $< 2\pi$ — teljesült-e. Ha nem, akkor a 17-es címkéjű utasítás FI értékéből 2π-t von le. Majd a vezérlés újra a részlet elejére

kerül. És ez így megy ciklikusan mindaddig, amíg a $\mathrm{FI} < 2\pi$ feltétel egyszer csak teljesül.

A részletben három olyan utasítás van, melyek FI értékétől függően többször végrehajtásra kerülhetnek. Ezek alkotják a *ciklust*. Ezek közül kettő „adminisztrál" és csupán a 17-es címkéjű utasítás végez tényleges kivonást. A továbbiakban azt fogjuk mondani, hogy ez az utasítás a *ciklus magjához* tartozik. Elképzelhető, hogy egy ciklusmag több utasításból áll. Ilyenre a következőkben sok példát adunk.

2. Írjunk FORTRAN programrészletet a négyzetgyökvonásra! Erre a célra alkalmazható a következő iterációs formula:

$$x_{k+1} = 0.5\left(x_k + \frac{A}{x_k}\right). \tag{3.37}$$

Kezdjük az iterációt az $x_0 = 1$ értékkel. Számláljuk az iterációk számát és ne engedjük, hogy az 10-nél több legyen!

Vegyük észre, hogy az ilyen típusú feladatokat programciklusok alkalmazásával kényelmes megoldani. Egy iterációs lépésnek a programciklus egyszeri „működése", lejátszása, lepörgetése felel meg. Valójában ritka a számológépen megoldott olyan feladat, mely utasítások egyszerű egymásutánjából áll, és nincs benne ciklus. A számológépek alkalmazása megkívánja a számítások ciklusos szervezését.

Mutassunk be egy lehetséges megoldást a példára:

```
      K=1
      X=1.0
15    XE=X
      X = 0.5*(X+A/X)                              (3.38)
      IF(ABS(1.0-X/XE)-EPS) 20,20,16
16    K = K+1
      IF(K-10) 15,15,17
      ⋮
```

A programrészlet minden egyes ciklusban megvizsgálja, hogy az utolsó két iteráció eredménye közti relatív eltérés abszolút értéke kisebb-e egy előre megadott EPS korlátnál. Ha kisebb, a számítás „bekonvergált", kellő pontossággal kapjuk a négyzetgyök értékét, és a vezérlés a program más részében elhelyezhető 20-as címkére adódik át. Ha viszont tíz iterációs ciklus nem volt elegendő ennek elérésére, a ciklus végén elhelyezett aritmetikai IF utasítás a vezérlést a 17-es címkére adja.

3. Írjunk FORTRAN programrészletet, mely a $ch(x)$ függvény értékét számítja ki 8 értékes számjegyre a függvény hatványsorának felhasználásával! Ez a sor:

$$y = ch(x) = 1 + \frac{x^2}{1\cdot 2} + \frac{x^4}{1\cdot 2\cdot 3\cdot 4} + \frac{x^6}{1\cdot 2\cdot 3\cdot 4\cdot 5\cdot 6} + \ldots \quad (3.39)$$

x-et adottnak tekintjük. Ha a hatványsor 10 tag figyelembevételével sem ad kívánt pontosságú eredményt, akkor adjuk a vezérlést a 20-as címkére. Ha az eredmény kellően pontos, azaz az eljárás, vagyis a függvénygenerálás konvergál, a vezérlést adjuk a 17-es címkére!

Egy megoldás:

```
      Y=1.0
      T=1.0
      Z = X**2
      AK=2.0
15    T = T*Z/(AK*(AK-1.0))                (3.40)
      Y = Y+T
      IF(T/Y-4E-9) 17,17,16
16    AK = AK+2.0
      IF(AK-19.0)15,20,20
      :
```

Ellenőrizzük programrészletünk működését! Az első négy utasítás a hatványsor első tagjának, 1-nek figyelembevételéről

gondoskodik: beállítja az első részletösszeget, Y = 1-et; beállítja az első tagot, T = 1-et; tekintettel arra, hogy a továbbiakban csak x^2 értékére lesz szükség, ezt elraktározza a Z változóba; végül AK-nak 2.0-t ad értékül, jelezve, hogy a soron következő tagban x második hatványa szerepel. A 15-ös címkéjű utasítással kezdődik a ciklus. Ez az utasítás a sor soronkövetkező tagját számítja ki az őt megelőző tag ismeretében. Ugyanis erre egyszerű formulát írhatunk fel (3.39) alapján:

$$T_i = T_{i-1} \frac{x^2}{2(i-1)(2i-3)}. \tag{3.41}$$

Az ilyen jellegű formulákat *rekurziós formuláknak* nevezzük. A bonyolultabb rekurziós formulák a sorozatból nem csak egyetlen megelőző tagot vesznek figyelembe, hanem esetleg jóval többet.

A következő utasítás a megfelelő részletösszeget képezi és helyezi el az Y változóba. Ezután azt kell ellenőrizni, hogy az utolsó tag elég kicsiny-e. A hibakorlátot 4.0×10^{-9}-re választottuk. A ciklus végén AK értékét növeljük meg a következő tagnak megfelelő kitevőértékre, végül megvizsgáljuk, hogy a következő tagban a kitevő nem magasabb-e, mint a megkívánt utolsó tagban fellépő 18.0 érték. Könnyen meggyőződhetünk arról, hogy a vezérlés a 20-as és 17-es címkékre a kívánalmaknak megfelelően adódik át.

4. A (3.23) programrészletben helyettesítsük a kijelölt GO TO utasítást számított GO TO utasítással! Fogalmazzuk át ennek megfelelően a részleteket!

Megoldás:

```
      :
   |KAPCS = 2
   |R = 15.0
   |GO TO 153
10 |BERUH1 = 48.6*F+125.4*T
   |R = 23.4
   |KAPCS = 3
```

```
        GO TO 153
11      BERUH2 = 81.3*F+42.8*T                  (3.42)
          :
          :
153     W = R**2
        F = 12.56637*W
        T = 4.18879*R*W
        GO TO(5,10,11,13,47), KAPCS
          :
```

Láthatjuk, hogy ebben az esetben az ASSIGN utasításokat közönséges értékadó utasítások helyettesítik.

5. Írjunk programrészletet, amely kiszámítja a

$$\mathrm{SUM} = \int_0^1 \frac{\mathrm{d}x}{1+x^2}$$

integrál értékét! Végezzük az integrálást a Simpson-formula szerint, a (0; 1) intervallumot 100 egyenlő részre osztva!

Mint a numerikus analízisből ismeretes, a Simpson-formula a következő alakú:

$$\int_a^b f(t)\,\mathrm{d}t \approx \frac{h}{3} \sum_{i=0}^{2n} c_i f_i \qquad (3.43)$$

ahol

$$h = \frac{b-a}{2n} \qquad f_i = f(a+i \cdot h)$$

$$c_i = \begin{cases} 1 \text{ ha } i = 0 \text{ vagy } i = 2n; \\ 2 \text{ ha } i \neq 0, i \neq 2n, \text{ de páros}; \\ 4 \text{ ha } i \text{ páratlan.} \end{cases}$$

A programrészletet összehasonlíthatóság kedvéért mind a kiszámított, mind a kijelölt GO TO utasítás felhasználásával

kidolgoztuk:

```
      SUM=0.0
      J=3
      I=1
13    X = FLOAT(I-1)/100.0
      F = 1.0/(1.0+X**2)
      GO TO(20,30,40),J
20    F = 4.0*F
      J=2
      GO TO 50
30    F = 2.0*F
40    J=1
50    SUM = SUM+F
      I = I+1
      IF(I-101)13,14,15
14    J=3
      GO TO 13
15    SUM = SUM/300.0
        :
```

```
      SUM=0.0
      ASSIGN 40 TO J
      I=1
13    X = FLOAT(I-1)/100.0
      F = 1.0/(1.0+X**2)
      GO TO J, (20,30,40)
20    F = 4.0*F
      ASSIGN 30 TO J
      GO TO 50
30    F = 2.0*F
40    ASSIGN 20 TO J
50    SUM = SUM+F
      I = I+1
      IF(I-101)13,14,15
14    ASSIGN 40 TO J
      GO TO 13
15    SUM = SUM/300.0
        :
```

(3.44)

Láthatjuk, hogy a J változó segítségével a speciális típusú GO TO utasítások a megfelelő c_i súlyfaktorokkal való szorzást adminisztrálják. Igaz, mivel itt csak háromirányú elágaztatásra van szükség, helyettük használhattuk volna az

```
      IF(J-2) 20,30,40
```

aritmetikai IF utasítást is a bal oldali programrészlet szerint. A programrészletek „működésének” megfejtését az olva-

sóra bízzuk. Megjegyezzük, hogy az integrál pontos értéke $\pi/4$, és száz részintervallumra való felosztás esetén ennek az értéknek igen nagy pontossággal kell adódnia.

4. Indexes változók, tömbök. Indexkifejezések

Sok matematikai feladatban nagyszámú, valami módon összetartozó mennyiség között kell hasonló műveleteket elvégeznünk. Gondoljunk például két többelemű vektor skaláris szorzatának kiszámítására:

$$T = \mathbf{u} \cdot \mathbf{v} = u_1 \cdot v_1 + u_2 \cdot v_2 + \cdots\cdots + u_{15} \cdot v_{15} \tag{3.45}$$

vagy két $n \times n$-es négyzetes mátrix összeszorzására:

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} \tag{3.46}$$

ami a fentiekhez hasonlóan komponensekben kiírva

$$c_{ik} = a_{i1} \cdot b_{1k} + a_{i2} \cdot b_{2k} + \cdots + a_{in} \cdot b_{nk}$$
$$(i, k = 1, 2, \ldots, n) \tag{3.47}$$

alakú. Ezeknek a műveleteknek a programozása eddigi eszközeinkkel nagyon nehézkes: minden vektorkomponens, minden mátrixelem számára külön jelölésre van szükség, minden egyes mátrixelemet külön utasítással kellene kiszámítani.

A vektorokkal és mátrixokkal való műveletek könnyen programozhatóak, ha felhasználjuk az *indexes változókat*, illetve a *tömböket*. A FORTRAN nyelvben a vektoroknak az ún. *egydimenziós tömbök*, a mátrixoknak a *kétdimenziós tömbök* feleltethetők meg. A tömböknek éppen úgy, mint a változóknak, azonosítóik vannak. A vektorok és mátrixok egyes komponenseinek, ill. elemeinek a *tömbelemek* vagy más szóval az *indexes változók* felelnek meg. Például:

FORTRAN jelölés: U(1), V(12), A(I,2), B(2,K), C(I,K)

jelentés: u_1, v_{12}, a_{i2}, b_{2k}, c_{ik}

Látjuk tehát, hogy a tömbelemeket vagy indexes változókat úgy jelöljük, hogy a tömbazonosítót zárójelpárba zárva követi az index, vagy követik — egymástól vesszővel elválasztva — az indexek. Az indexek mindig egész típusú mennyiségek. Az indexek száma a FORTRAN nyelvben maximálisan három,

azaz a FORTRAN-ban maximálisan három dimenziós tömbök vannak értelmezve. (Egyes gépi reprezentációkban az indexek száma lényegesen nagyobb is lehet.) A tömbök méreteit, azaz a tömbök elemeinek számát összesen és az egyes dimenziókban véve a FORTRAN nyelv nem korlátozza. Tehát a FORTRAN kezelni tudja mind egy százkomponensű vektor, mind egy 8×8-as mátrix megfelelőit: a százelemű egydimenziós és a 8×8-as kétdimenziós tömböt.

Ezzel a jelöléstechnikával a (3.45) vektorszorzást és a (3.47) mátrixszorzást a következő programrészletekkel lehet elvégezni:

```
      T=0.0
      J=1
   18 T = T+U(J)*V(J)                 (3.48)
      J = J+1
      IF(J-15) 18,18,20
   20  {folytatás}
```

```
      I=1
   16 K=1
   17 T=0.0
      J=1
   18 T = T+A(I,J)*B(J,K)
      J = J+1                         (3.49)
      IF(J-N)18,18,20
   20 C(I,K)=T
      K = K+1
      IF(K-N)17,17,21
   21 I = I+1
      IF(I-N)16,16,22
   22  {folytatás}
```

Egy kissé elemezzük előbb (3.48), majd (3.49) működését. Az előbbiben végeredményben csak a 18-as címkéjű utasítás „dolgozik": T-hez hozzáadja az u és v vektorok J-edik komponenseinek szorzatát. Ha minden J-re elvégezzük ezt a hozzáadást és eredetileg T értéke nulla volt, akkor végül T-ben valóban a vektorok skaláris szorzata képződik. A többi utasítás különféle adminisztrációs tevékenységeket végez: beállítja T kezdeti nulla értékét, 1-et ad értékül J-nek, minden ciklusmenetben megnöveli 1-gyel J értékét, végül ellenőrzi, hogy J értéke nem haladta-e túl a 15-öt. Ugyanezt teszi a (3.49) programrészlet „magja". Figyeljük meg a 17-es címkéjű utasítástól a 20-as címkéig terjedő programrészletet! Csupán abban tér el (3.48)-tól, hogy 15 helyett N áll és a záróutasításban C(I, K) átveszi T értékét. A további utasítások csak azt adminisztrálják, hogy I és K 1-től N-ig minden kombinációban felvegye a megfelelő értékeket, tehát az összes C(I, K) sorra kerüljön.

Láthatjuk, hogy a (3.48) programrészlet J különféle értékeire egymásután elvégzi a 18-as címkéjű utasítást. Az ilyen programműködést *programciklusnak* (huroknak) nevezzük. A (3.48) programrészletben a J változó értéke lépésenként egyenletesen változik. Az ilyen egész típusú változót a továbbiakban *ciklusváltozónak* nevezzük. A 18-as címkéjű utasítás a *ciklusmag*, mely jelen esetben egyetlen utasításból áll.

A (3.49) programrészletben hasonló ciklust találunk a J ciklusváltozóval, azonban ezt körülöleli egy másik ciklus a K ciklusváltozóval. Ennek a másik ciklusnak a magja a 17-es címkéjű értékadó utasításból, a belső ciklusból és egy értékadásból áll. Legkívül ismét ciklust találunk az I ciklusváltozóval, melynek magja a K ciklusváltozójú ciklus.

Az ilyen programszerkezetet *egymásba skatulyázott ciklusoknak* nevezzük. (A ciklusok kényelmesebb szerkesztésére külön utasítás, a ciklusutasítás is használható, melyet a következő pontban ismertetünk.)

Példáink azt mutatják, hogy az indexes változók használata nagyon sokszor célszerű. Ismerkedjünk meg ezért az erre vonatkozó főbb szabályokkal:

1. A tömbök méreteit (a dimenziók számát, valamint a komponensek számát minden egyes dimenzió „irányában") mindig deklarálni kell (lásd a Negyedik fejezet 3. pontját).

2. Tetszőleges típusú tömböt használhatunk programja-

inkban, még logikai tömböket is. Ha külön nem deklaráljuk a tömbök típusát, a fordítóprogram a tömböt a tömbazonosító első betűje alapján az automatikus típusdeklaráció szerint valós vagy egész típusúnak tekinti.

3. Tömbazonosító önmagában indexek nélkül csak bizonyos esetekben használható a programban; operandusként és értékadó utasítás bal oldalán nem. Hasonlóképpen nem lehet változóra és tömbre ugyanazzal az azonosítóval hivatkozni.

4. Az ASA FORTRAN-ban tömbök indexei csak ún. *standard indexkifejezések* lehetnek.

A standard indexkifejezések a következő egész kifejezés típusokból kerülhetnek ki (a továbbiakban *c* és *k* pozitív egész számokat, I egész változót jelöl):

típusok	*példák*
c ∗ I + k	2 ∗ I + 5
c ∗ I − k	8 ∗ J − 3
c ∗ I	2 ∗ K
I + k	L + 2
I − k	M − 1
I	N
k	4

Ahhoz, hogy egy tömbelemmel számolni lehessen, az kell, hogy a megfelelő indexkifejezéseknek — így a bennük szereplő egész változóknak — értékük legyen. Továbbá az is szükséges, hogy az indexek száma megegyezzék a tömb deklarált dimenziójával, és az indexek aktuális értéke olyan pozitív egész szám legyen, mely nem haladja meg a tömbnek a megfelelő dimenzió „irányában” deklarált méretét.

A számológép tehát a megfelelő tömbelemmel csak úgy tud dolgozni, hogy előbb kiértékeli az indexeket. Ebből következik, hogy bármiféle számolás mindig lassúbb indexes változóval, mint ha az indexes változó helyett közönséges változó szerepel. Így érthető, miért használtuk a (3.49) programrészletben is a T változót részleteredmények gyűjtésére és miért külön utasítással adtuk át C(I, K)-nak a megfelelő végeredményt.

A standard indexkifejezésekben nemcsak a számok és a változó típusa és értéke speciális, hanem az egyes „alkotórészek" sorrendje is lényeges. Ezek szerint pl. nem standard indexkifejezések a következők: 5+I 6−J −K+5.

Egyes gépi reprezentációkban a standard indexkifejezések helyett tetszőleges egész kifejezések használhatók. Szokásos megkötés viszont, hogy ezekben az egész kifejezésekben indexes változók és függvénykifejezések ne szerepeljenek.

5. Ciklusutasítások és használatuk

a) A DO utasítás és a CONTINUE utasítás

Az indexes változókkal kapcsolatban szó esett már a számítási ciklusokról, a ciklusváltozóról és a ciklus magjáról is. Emeljük most ki a (3.49) programrészletből a K ciklusváltozóra vonatkozó ciklust, és nézzük meg ennek a ciklusnak a szervezését:

```
16   K=1
17   T=0.0
     J=1
18   T = T+A(I, J) * B(J, K)
     J = J+1
     IF(J-N) 18, 18, 20
20   C(I, K)=T
     K = K+1
     IF(K-N)17,17,21
21
```

(3.50)

Látható, hogy a ciklus szervezése a következő adminisztrációt követeli meg:

1. A ciklusváltozó felvesz valamilyen *kezdőértéket* (esetünkben a K ciklusváltozó 1-et).

2. Végrehajtódik valamennyi ciklusmaghoz tartozó utasítás (esetünkben az utolsó a 20-as címkéjű).

3. A ciklusváltozó értéke megnövelődik valamilyen *növekményértékkel* (esetünkben 1-gyel).

4. A program megvizsgálja, hogy a ciklusváltozó nem haladott-e meg valamilyen *végértéket* (esetünkben N-et).

5. Ha a ciklusváltozó értéke még nem haladta meg a végértéket, a vezérlés újra a ciklusmag első utasítására adódik vissza.

6. Ha a ciklusváltozó meghaladta a végértéket, a vezérlés a ciklust követő első utasításra adódik át.

Tekintettel arra, hogy a ciklusok ilyen szervezése a programozói gyakorlatban rendkívül gyakori, a FORTRAN nyelv a ciklusok megadására összevont jelölést, külön utasítást vezetett be. Például a fenti (3.50) ciklus a következő egyszerűsített jelöléssel is megadható:

```
      DO 20  K=1,N
      T=0.0
      J=1
18    T = T+A(I,J)*B(J,K)        ciklusmag
      J = J+1
      IF(J-N) 18,18,20
20    C(I,K)=T
```
(3.51)

azaz: végezd el a 20-as címkéig felsorolt utasításokat a K ciklusváltozó értékeire rendre, míg a ciklusváltozó 1-től N-ig növekszik (egyesével), majd térj rá a következő utasításra! Az utasítás hatására a gép a korábban leírt tevékenységet végzi el. Tehát ez a ciklusutasítás a (3.50) programrészlet 3 utasítását helyettesíti.

Adjuk meg most pontosan a ciklusutasításra vonatkozó szabályokat.

A FORTRAN nyelvben a ciklusutasítás általános alakja a következő:

$$\text{DO}\ \ l\ \ i=m_1, m_2, m_3 \tag{3.52}$$

vagy

$$\text{DO}\ \ l\ \ i=m_1, m_2 \tag{3.53}$$

ahol

1. a DO *alapszó*, amely az utasítás elnevezésére is szolgál.

2. l a ciklusmag utolsó utasításának címkéje. Ez az utasítás felírási sorrendben valahol a DO utasítás után kell, hogy szerepeljen és általában nem lehet vezérlő utasítás.

3. Az

$$i=m_1, m_2, m_3$$

és az

$$i=m_1, m_2$$

szerkezeteket *ciklusspecifikációknak* nevezzük. (Alkotórészeik tárgyalását lásd alább!) A ciklusspecifikáció m_1-től m_2-ig m_3-anként rendre értéket ad a ciklusváltozónak a ciklusmag minden egyes lefutása előtt, illetve, ha m_3 nem szerepel, a ciklusspecifikáció szerkezet hatása megegyezik az

$$i=m_1, m_2, 1$$

ciklusspecifikáció hatásával.

Hasonló ciklusspecifikációk a DO utasításon kívül még az ún. összetett I/O lista elemben is (lásd a Hatodik fejezet 2. pontját) előfordulnak.

4. i egész típusú változót, a *ciklusváltozót* jelöli. Mivel a ciklusspecifikáció értéket ad a ciklusváltozónak, a ciklusváltozó korábbi értéke (ha volt) a ciklus végrehajtásának megkezdésekor elvész.

Amíg a ciklus dolgozik, „jár", tilos minden olyan értékadó utasítás alkalmazása, mely befolyással lenne a ciklusváltozó értékére. Ha a ciklus normális úton végetér, azaz „lejár", ciklusváltozója értékét az ASA FORTRAN nem határozza meg.

Ilyen módon az ASA FORTRAN szabad kezet ad a fordítóprogramok íróinak abban, hogy milyen értéket hagyjanak a ciklusváltozóban a ciklus végén. Egyes gépi reprezentációkban ez azt jelenti, hogy a ciklusváltozó értéke a ciklus „lejárása" után az az első megnövelt érték lesz. amivel a ciklus már nem futott le.

5. m_1-et *kezdőparaméternek*, m_2-t *végparaméternek* és m_3-at *növekmény-paraméternek* nevezzük. Ezek egész konstansok, vagy egész típusú változók lehetnek. Értéküket a DO ciklus végrehajtása során semmilyen módon nem szabad megváltoztatni. A DO ciklus megkezdésekor — úgyis mondjuk, hogy a DO ciklusba való belépéskor — értékük pozitív kell hogy legyen, azonban nem követeljük meg, hogy $m_2 \geqq m_1$ fenn-

álljon. $m_2 < m_1$ esetben a ciklus rendhagyó módon dolgozik (lásd 77. oldal!). Ha m_3 nem szerepel, értéke 1.

Egyes gépi reprezentációkban m_1, m_2 és m_3 pozitív aktuális értékű és egész tipusú kifejezés is lehet.

Mielőtt példákkal illusztrálnánk a DO ciklusutasítás használatát, vezessük be a FORTRAN nyelv legegyszerűbb utasítását, az üres utasítást, melynek egyedüli szerepe az, hogy címkét viselhet:

```
16    CONTINUE
                                        (3.54)
50    CONTINUE
```

Funkciója éppen az, hogy nem csinál semmit. Leggyakrabban a DO ciklusok záró, vagy utolsó utasításaként fordul elő, bár nem mindig kötelező ezen a helyen a CONTINUE utasítás használata.

Ezek után példákként fogalmazzunk át néhány korábban bemutatott programrészletet a DO utasítás felhasználásával!

1. A (3.38) négyzetgyökvonó programrészlet egy megfelelője a következő:

```
      X=1.0
      DO 16 K=1,10
      XE=X
      X= 0.5*(X+A/X)                    (3.55)
      IF(ABS(1.0-X/XE)-EPS) 20,20,16
16    CONTINUE
17    :
```

Ebből látható, hogy a ciklusmag utolsó utasítása is viselhet olyan címkét, melyre ugró utasítás hivatkozik (lásd pontosabban a következő pontot!), sőt esetünkben az IF utasításban szereplő 20-as címke a cikluson kívül van. Igaz tehát, hogy ciklusból szabad kiugrani; de itt az alkalom annak megemlítésére, hogy fordított irányú — tehát befelé — ugrás még a ciklus

címkézett utolsó utasítására is általában tilos! Erről bővebbet a *c)* pontban mondunk!

Ha ilyen rendhagyó módon hagyja el a program a ciklust, anélkül, hogy az végig „lejárt" volna, a ciklusváltozó a cikluson kívülre is magával viszi utolsó aktuális értékét.

2. Tekintsük a (3.40) ch (x) „függvénygenerátor" programrészlet megfelelőjét, ciklusutasítással felírva:

```
      Y=1.0
      Z = X**2
      T=1.0
      DO 16 K=2,19,2
      AK=FLOAT(K)                              (3.56)
      T = T*Z/(AK*(AK-1.0))
      Y = Y+T
      IF(T/Y-4E-9) 20,20,16
16    CONTINUE
```

Mint látható, most nehézséget okozott volna, hogy a ciklusváltozó csak egész típusú lehet. Ezt a nehézséget a FLOAT belső standard függvény alkalmazásával hidaltuk át.*

3. Írjunk programot a (3.48) vektorszorzásra a DO utasítás felhasználásával:

```
      T=0.0
      DO 18 J=1,15
18    T = T+U(J)*V(J)                          (3.57)
      :
```

* Történetesen itt az AK=K utasítás is megfelelt volna, mint ahogy a 6. táblázat (az 50 oldalon) ezt megengedi, de sok esetben az ún. konverziós függvények alkalmazása a legegyszerűbb út.

4. Írjunk programot a (3.49) mátrixszorzásra:

```
      DO 21 I=1,N
      DO 20 K=1,N
      T=0.0
      DO 18 J=1,N                          (3.58)
18    T = T+A(I,J)*B(J,K)
20    C(I,K)=T
21    CONTINUE
```

Láthatjuk, hogy a külső ciklusok magjában a „beskatulyázott” ciklusok teljes egészükben benne vannak.

Nézzünk most egy-két további példát:

5. Számítsuk ki a

$$p = \sum_{i=1}^{N} a_i x^{(N-i)} \tag{3.59}$$

polinom értékét a Horner-séma szerint! Megjegyezzük, hogy a polinom értékét így számíthatjuk ki a legkevesebb művelet felhasználásával. A Horner-séma alakja:

$$(\ldots((a_1 x + a_2)x + a_3)x + \ldots)x + a_N \tag{3.60}$$

A megoldás:

```
      P=A(1)
      DO 15 I=2,N                          (3.61)
15    P = P*X+A(I)
```

Megjegyezzük, hogy ez a programrészlet N=1 esetén hibásan működik.

6. Tabellázzuk az

$$f(t) = A_1 + A_2 e^{-A_3 t} \tag{3.62}$$

függvényt a $0.1 < t \leqq 1$ intervallumban tizedenként, az $1 < t \leqq 10$

intervallumban egészenként és végül a $10 < t \leqq 100$ intervallumban 10-esével! Két lehetséges megoldást mutatunk be:

a)

```
      I = 1
      Z = 0.1
      DO 15 J = 1,3
      DO 14 J1 = 2,10
      T = FLOAT (J1)*Z                                    (3.63)
      F(I) = A1+A2*EXP(—A3*T)
14    I = I+1
15    Z = Z*10.0
```

b)

```
      I = 1
      Z = EXP(—0.1*A3)
      A2 = A2*Z
      DO 15 J = 1,3
      DO 14 J1 = 2,10                                     (3.64)
      A2 = A2*Z
      F(I) = A1+A2
14    I = I+1
15    Z = Z**10
```

Ha összehasonlítjuk a két megoldást, megállapítható, hogy az *a)* változat működése világos, viszont minden iterációs ciklusban felhasználja a FLOAT és az EXP standard függvényeket. Ez nagyon lelassítja a programrészlet működését, ezzel szemben a táblázat pontos. A *b)* változat rendkívül gyors, egyetlenegyszer hívja az EXP külső standard függvényt, utasításai is megmagyarázhatóak, ha figyelembe vesszük az

$$e^{-x-y} = e^{-x} \cdot e^{-y}$$

azonosságot. Ezzel szemben pontatlan és „elrontja" az A2 paraméter értékét. Ez azt jelenti, hogy a programrészlet végrehajtása után A2 értéke nem a megadott paraméterérték lesz, hanem körülbelül $A2 \cdot e^{-100 \cdot A3}$. Tehát ha ezt a részletet követőleg szükség lenne A2 értékére, akkor azt előzőleg „ki kell menteni", pl. a következő módon:

```
      OVOLUK = A2
      I = 1
      Z = EXP(-0.1*A3)
      A2 = A2*Z
      DO 15 J = 1,3
      DO 14 J1 = 2,10
      A2 = A2*Z
      F(I) = A1+A2
14    I = I+1
15    Z = Z**10
      A2 = OVOLUK
```
(3.65)

7. Állapítsuk meg, hogyan működik a

```
      P = A(1)
      Z = 2.0
      DO 15 J = 3,N
      Z = Z**2
15    P = P*Z+A(J)
      :
```

programrészlet N különböző értékeire! Legyen N = 3, 4, 5, 2, 1! Vegyük figyelembe a (3.50) értelmezést az őt követő magyarázattal!

Megoldás: Z és P értékei a ciklus elvégzése után, N különböző értékeire a következők:

N	Z	P
3	4·0	$4a_1+a_3$
4	16·0	$16(4a_1+a_3)+a_4$
5	256·0	$256(16(4a_1+a_3)+a_4)+a_5$
2	4·0	$4a_1+a_3$
1	4·0	$4a_1+a_3$

Nem hiba! Valóban, a DO ciklus szervezése olyan, hogy a ciklusmag lefutása után vizsgálja meg, vajon a ciklusváltozó értéke meghaladta-e a végparaméter értékét. Ezért *a ciklus magja legalább egyszer lefut, ha a vezérlés a* DO *utasításra került.*

b) Ciklusok egymásba skatulyázása

Mint ahogyan a (3.49), (3.58) és (3.63) példákban már láttuk, a programozás során szükség lehet két vagy több ciklus egymásba skatulyázására. Ezen azt értjük, hogy a ciklusok egymáshoz viszonyítva úgy viselkednek, mint az aritmetikai kifejezésekben a zárójelpárok. Tehát egy ciklus akkor és csakis akkor van „beskatulyázva" egy külső — vagy külsőbb — ciklusba, ha minden utasítása hozzátartozik a külső ciklus magjához.

Egymásba skatulyázott ciklusok használatával kapcsolatban néhány szabályra külön fel kell hívni a figyelmet. Ezeket egyetlen „külső" és egyetlen „belső" ciklus viszonylatában teszszük szemléletessé:

1. Az előzőleg már mondottak szerint a belső ciklus minden utasítása a külső ciklus ciklusmagjához kell, hogy tartozzék. Gyakori eset, hogy két egymásba skatulyázott ciklusnak közös záróutasítása van. Ekkor a két DO utasítás ugyanarra a címkére hivatkozik. Ilyen esetben *a végső címkére csak a belső ciklusban elhelyezett vezérlőutasítással szabad ráugrani.* (Lásd a pont végén levő példákat.)

2. Ismeretes, hogy ciklus belsejében nem szabad a ciklusváltozónak értéket adni. Ha a külső és a belső ciklus ciklusváltozója ugyanaz az egész változó lenne, akkor a belső ciklus specifikációja a külső ciklus minden egyes lefutása során értékeket adna a közös ciklusváltozónak, így megzavarná a külső ciklus működését. Kimondhatjuk tehát:

Ciklus belsejében nem fordulhat elő olyan ciklusspecifikáció — sem a beleskatulyázott ciklus DO *utasításában, sem összetett* I/O *listaelemben* (lásd a Hatodik fejezet 2. pontját)—, *melynek ciklusváltozója azonos a külső ciklus ciklusváltozójával.*

Ennek a szabálynak megfelelően az egymásba skatulyázott ciklusok ciklusváltozóinak mind különbözniök kell egymástól.

Egyes gépi reprezentációk fordítóprogramjai nem tekintik hibásnak az olyan programokat, melyekben pl. a ciklusváltozó új értéket kap a ciklus belsejében. Ilyen esetben előfordulhat azonos ciklusváltozó két egymásba skatulyázott „hatáskörrel" rendelkező ciklusspecifikációban is. Azt, hogy ilyenkor a célprogram hogyan működik, csak a fordítóprogram ismeretében lehet megmondani. Ezért ez a tulajdonság sok programhibát tesz lehetővé.

Példák

1. Transzponáljuk az $N \times N$-es A mátrixot a főátlójára! Legyen az A mátrix az A(N, N) tömbben elhelyezve; a transzponált A^t mátrix keletkezzék ugyanebben a tömbben. A transzponált mátrix definíciója — mint ismeretes —

$$\|a^t_{ik}\| = \|a_{ki}\|$$

Egy megoldás:

```
      M = N-1
      DO 20 K = 1,M
      M1 = K+1
      DO 20 K1 = M1,N
      B = A(K,K1)
      A(K,K1) = A(K1,K)
   20 A(K1,K) = B
```

(3.66)

Két egymásba skatulyázott ciklussal letapogattuk a mátrix főátló feletti elemeit és kicseréltük őket a főátló alatti megfelelő elemekkel.

2. Bontsuk fel az $N \times N$-es **A** mátrixot az **S** szimmetrikus és a **P** antiszimmetrikus részmátrixokra! Ezekre fennállnak:

$$\mathbf{A} = \mathbf{S} + \mathbf{P}$$

$$\mathbf{S} = \frac{1}{2}[\mathbf{A} + \mathbf{A}^t] \qquad \mathbf{P} = \frac{1}{2}[\mathbf{A} - \mathbf{A}^t]$$

$$s_{ik} = s_{ki} = \frac{1}{2}[a_{ik} + a_{ki}] \qquad p_{ik} = -p_{ki} = \frac{1}{2}[a_{ik} - a_{ki}]$$

$$(i, k = 1, 2, \ldots, N)$$

Könnyű felismerni, hogy a főátlóban — ahol $k = i$ — a **P** mátrixban nullák állnak, az **S** mátrixban pedig az **A** mátrix főátlójának elemei. Mivel az **S** mátrix szimmetrikus a főátlóra, tehát a főátló alatti elemeit nem kell külön feljegyezni, hiszen azok már úgyis szerepelnek a főátló felett a megfelelő helyen. Hasonlóképpen a **P** mátrixot teljesen megadtuk, ha a főátló alatti elemeit megadjuk (vagy fordítva). Ezért helymegtakarítás céljából állítsuk elő az **S** mátrix elemeit az **A** mátrix „felső felében" a főátlót is beleértve, a **P** mátrix elemeit pedig az **A** mátrix „alsó felében".

Egy lehetséges megoldás:

```
      N1 = N-1
      DO 3 I=1,N1
      I1 = I+1
      DO 3 K=I1, N
      S = 0.5*(A(I,K)+A(K,I))                    (3.67)
      A(K,I) = A(K,I)-S
3     A(I,K)=S
      :
```

Ez a programrészlet hibásan működik, ha $N=1$. Különben végigfut a főátló fölötti mátrixelemeken és minden elem kapcsán a szimmetrikus elemre is elvégzi a szükséges számításokat. Látható, hogy az adott A mátrixon kívül alig használ más változót és nem használ más tömböt. Így helyigénye rendkívül kicsi. A ciklusmag úgy van kidolgozva, hogy lehetőleg kevésszer kerüljenek az indexek kiértékelésre. Így ez a programrészlet a FORTRAN nyelven írható hasonló célú programrészletek közül elég gyors.

3. Legyen V egy N-elemű vektor. Írjunk programrészleteket, melyek

a) megadják a maximális és minimális elem számértékét;

b) megadják a maximális és minimális elem számértékét és indexét, ha nincsenek egyenlőek a vektorelemek között;

c) nagyság szerint csökkenő sorba rendezik az elemeket;

d) abszolút értékeik szerint sorba rendezik az elemeket.

Lehetséges megoldások:

a)

```
      OMAX=V(1)
      OMIN=V(1)
      DO 1 I=2,N                         (3.68)
      OMAX=AMAX1(OMAX,V(I))
    1 OMIN=AMIN1(OMIN,V(I))
```

A megoldás $N=1$-re nem működik. Belső standard függvényeket használtunk fel a megoldásban.

b)

```
      OMAX=V(1)
      OMIN=V(1)
      IMAX=1
      IMIN=1
      DO 1 I=2,N
      IF(OMAX-V(I)) 2,3,3
    2 OMAX=V(I)                          (3.69)
```

```
      IMAX=I
      GO TO 1
 3    IF(OMIN-V(I)) 1,1,4
 4    OMIN=V(I)
      IMIN=I
 1    CONTINUE
```

A megoldásban kihasználtuk azt a tényt, hogy ha egy elem maximálisnak bizonyul, már nem bizonyulhat minimálisnak is.

c) Most járjunk el úgy, hogy kiválasztjuk a legnagyobb elemet és azt hozzuk az első helyre, majd a maradék vektorra megismételjük az eljárást, és ezt mindaddig folytatjuk, míg csak a maradék vektor egyelemű nem lesz:

```
      N1 = N-1
      DO 1 I=1,N1
      K=I
      K1 = I+1
      OMAX=V(I)
      DO 2 J=K1,N
      IF(OMAX-V(J)) 4,2,2
 4    OMAX=V(J)                          (3.70)
      K=J
 2    CONTINUE
      IF(K-I) 29,1,3
 3    V(K)=V(I)
      V(I)=OMAX
 1    CONTINUE
```

d) A megoldás módszere azonos az előzővel, de a vizsgálatokat V(I) abszolút értékére kell végezni:

```
      N1 = N-1
      DO 1 I=1,N1
      K=I
      K1 = I+1
      OMAX=ABS(V(I))
      DO 2 J=K1,N
      DOLG=ABS(V(J))
      IF(OMAX-DOLG) 4,2,2
4     OMAX=DOLG
      K=J
2     CONTINUE
      IF(K-I) 29,1,3
3     DOLG=V(I)
      V(I)=V(K)
      V(K)=DOLG
1     CONTINUE
```

(3.71)

4. Folyadékok turbulens áramlására sima felületű csövekben fennáll az

$$\frac{1}{\sqrt{\lambda}} = 2\lg(R\sqrt{\lambda}) - 0.8 \tag{3.72}$$

összefüggés, ahol R az ún. Reynolds-szám és λ a cső ellenállástényezője. Írjunk programrészletet, mely R ismeretében kiszámítja λ értékét!

A feladatot célszerűen iterációval oldhatjuk meg. Vezessük

be az $u = 1/\sqrt{\lambda}$ változót, melynek segítségével összefüggésünk az

$$u = 2 \cdot \lg\left(\frac{R}{u}\right) - 0.8 \tag{3.73}$$

alakban írható. Ha u helyére a jobb oldalon u_k-t, a bal oldalon u_{k+1}-et írunk, előttünk áll az iterációra alkalmas kifejezés:

$$u_{k+1} = 2 \cdot \lg\left(\frac{R}{u_k}\right) - 0.8 \tag{3.74}$$

melynek segítségével az u változó k-adik közelítéséből, u_k-ból a $k+1$-edik közelítés számítható. Az iteráció elegendően nagy u, tehát elég kis λ esetén konvergál (a gyakorlati esetekben ez a feltétel teljesül).

Egy lehetséges megoldás, ha az u_0 kiinduló értéket 7-nek választjuk és az iterációk maximális számát pedig 10-nek:

```
      U=7.0
      DO 1 I=1,10
      UE=U
      U = 2.0*ALOG10(R/U)-0.8
      IF(ABS(U-UE)-1E-4) 2,2,1                    (3.75)
2     ALAMBD = 1.0/U**2
      GO TO 18
1     CONTINUE
20    :
```

A programrészletben egy iterációs ciklus lefutásának a DO ciklus magjának egyszeri lefutása felel meg. Ha az iteráció 10 lépésben nem konvergált be, a vezérlés a 20-as címkére, sikeres iteráció esetén pedig a 18-as címkére kerül.

5. Írjunk programrészletet az N-ismeretlenes inhomogén lineáris algebrai egyenletrendszer megoldására!

Az egyenletrendszer:

$$\begin{aligned} a_{11}x_1+a_{12}x_2+\dots+a_{1N}x_N &= a_{1\,N+1}\\ a_{21}x_1+a_{22}x_2+\dots+a_{2N}x_N &= a_{2\,N+1}\\ &\dots\\ a_{N1}x_1+a_{N2}x_2+\dots+a_{NN}x_N &= a_{N\,N+1} \end{aligned} \tag{3.76}$$

Tegyük fel, hogy (3.76) bal oldali együtthatómátrixának determinánsa, valamint sarokdeterminánsainak sorozata:

$$a_{11}, \quad \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}, \quad \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}, \text{ stb.} \tag{3.77}$$

nullától különbözik, illetve nincsen köztük kis abszolút értékű tag. Ebben az esetben például a *Gauss—Jordan*-féle eliminációs algoritmus a legegyszerűbb alakjában alkalmazható a feladat megoldására. Bár műveletigényes eljárás és ezért a gyakorlatban nem okvetlenül használandó, szemléletessége és könynyű programozhatósága miatt válasszuk most ezt az algoritmust, mely ebben az esetben a következőképpen végezhető:

Az I = 1, 2, ..., N-re rendre végrehajtjuk a következőket:

a) Az I-edik sor I+1, I+2, ..., N+1-edik elemét elosztjuk a_{II}-vel. Ez az osztás mindig elvégezhető, ha a (3.77) determinánsok nem nullák.

b) A J = 1, 2, ..., I−1, I+1, ..., N-edik sorokban a K = I+1, I+2, ..., N+1-edik elemekből rendre levonjuk az I-edik sor K-adik elemének a_{JI}-szeresét.

Ha *a)*-t és *b)*-t I összes értékére rendre végrehajtottuk, az egyenletrendszer megoldását meg fogjuk kapni az egyenletrendszer jobb oldali változóiban!

Az algoritmus egy lehetséges felírási módja:

```
N1 = N+1
DO 10 I=1,N
Q=1.0/A(I,I)
I1 = I+1
DO 2 K=I1,N1
```

```
2      A(I,K) = Q*A(I,K)
       DO 10 J=1,N                     (3.78)
       IF(J-I) 3,10,3
3      T=A(J,I)
       DO 9 K=I1,N1
9      A(J,K) = A(J,K)-T*A(I,K)
10     CONTINUE
```

A részlet működésének ellenőrzését az olvasóra bízzuk. Megjegyezzük, hogy az egymásba skatulyázott DO-ciklusok közül a 9-es zárócímkéjűt nem lehetett a külső ciklusokkal közös utasításon befejezni, mert egy aritmetikai IF utasítás ekkor e legbelső ciklushoz viszonyítva „kívülről" ugrana a zárócímkére.

Ha a (3.77) determinánsok közül egy-kettő nulla körüli értékű, akkor az algoritmus csak bizonyos módosításokkal használható. Ilyenkor alkalmazhatók pl. az ún. főelemkiválasztó algoritmusok, amelyeknek tárgyalására nem térünk ki.

Ha tüzetesebben megvizsgáljuk a (3.78) programrészlet működését, észrevehetjük, hogy a (3.76) egyenlet jobb oldali változói csupán a 2-es és a 9-es címkéjű utasításban szerepelnek, mégpedig abban az esetben, amikor K éppen az N1 = N+1 értéket veszi fel. Ily módon ezeknek nincs hatása a programrészlet többi változójának értékére, vagyis az egyenletrendszer bal oldali együtthatóinak, valamint a T és Q segédváltozóknak értékeire. Ez a tény kihasználható arra, hogy tetszőleges számú különböző jobb oldalakra egyszerre oldjunk meg több egyenletet, feltéve, hogy a bal oldali együtthatók azonosak. Legyen a különböző jobb oldalak száma M. Ebben az esetben algoritmusunk csupán annyiban módosul, hogy első utasítása, N1 = N+1 helyett az N1 = N+M utasítást kell beírnunk.

A most körvonalazott „bővítés" lényegében az

$$\mathbf{A} * \mathbf{X} = \mathbf{B} \tag{3.79}$$

mátrixegyenlet megoldását jelenti az **X** mátrixra, ahol **A** $N \times N$-es mátrix, **X** és **B** pedig $N \times M$-es mátrixok. A **B** mátrixot az **A** mátrix végéhez kell „hozzáfűzni".

6. A (3.79) mátrixegyenlet speciális esete a mátrixinverzió, melyben **B** éppen az N×N-es egységmátrix, melynek csak a főátlóban levő elemei különböznek nullától, ugyanis éppen egységnyiek. Írjunk olyan programrészletet, mely invertálja az N×N-es **A** mátrixot és amelynek helyigénye kicsi!

Egy lehetséges megoldás lenne a (3.78) programrészlet, ha első sorában az N1 = N+1 utasítás helyett N1 = 2*N-et írnánk és előzőleg a **A** mátrix hátsó feléhez hozzáfűznénk az egységmátrixot, pl. a

```
      DO 1 J=1,N
      N1=N+J
      DO 4 L=1,N
      L1=L+N
4     A(J,L1) = 0.0
1     A(J,N1) = 1.0
```
(3.80)

programrészlettel. Igenám, de akkor az eljárás helyigénye nagyobb, mint $2N^2$, hiszen az eredeti **A** mátrix az egységmátrixszal együtt foglal helyet a számológép memóriájában. Ha azonban jobban megfigyeljük eljárásunk működését, észrevehetjük, hogy az I ciklus lefutásai során rendre feleslegesekké és fel nem használttá válnak az **A** mátrix oszlopai a legelső oszloptól kezdve. Ugyanakkor az egységmátrix oszlopai pontosan ugyanilyen ütemben telnek be. Felmerül az ötlet: írjuk be ezeket az oszlopokat azonnal az **A** mátrix további felhasználásra nem kerülő első oszlopaiba. Ilyen elven működő programrészlet:

```
      DO 10 I=1,N
      W=A(I,I)
      Q=1.0/W
      DO 2 K=1,N
      A(I,K) = Q*A(I,K)
      IF(K-I) 2,1,2
1     A(I,K)=Q
```

```
2     CONTINUE
      DO 10 J=1,N
      IF(J-I) 3,10,3
3     T=A(J,I)
      Q1=-T/W
      DO 9 K=1,N
      A(J,K)=A(J,K)-T*A(I,K)
      IF(K-I) 9,4,9
4     A(J,K)=Q1
9     CONTINUE
10    CONTINUE
```

(3.81)

c) Kiterjesztett hatáskörű ciklus

Az eddigiek alapján az olvasónak szemléletes képe van a programciklusról, így a ciklusmag fogalmáról is. Utóbbi a következőképpen definiálható: ciklusmagon azoknak az utasításoknak az összességét értjük, melyeket a ciklus lefutása során a program megismétel, illetve megismételhet; nem tartoznak viszont a ciklusmaghoz a ciklusszervezés adminisztratív utasításai.

Az előzőkben megismerkedtünk a DO utasítással, mint a FORTRAN nyelv ciklusutasításával. Láttuk, hogy ciklusok programozásának eléggé kényelmes, bár nem kizárólagos eszköze, hiszen nélküle is bármely ciklus programozható, sőt speciális esetekben előfordulhat, hogy a DO utasítást célszerű mellőzni. Jelen alpontban a ciklusmag és a DO utasítás kapcsolatát fogjuk pontosabban elemezni.

A ciklusparaméter beállításához és ellenőrzéséhez szükséges információkat a DO utasítás ciklusspecifikációjában vagy ún. ciklusspecifikációs részében lehet megadni. A DO utasításban szereplő címke a ciklusmag utolsó vagy záró utasítását adja meg. Eddigi ismereteink alapján még nem dönthető el, hogy ennek az utasításnak utolsó volta az utasítások *végrehajtásának* vagy *felsorolásának* sorrendjében értendő-e. Eddigi példáinkban minden esetben a ciklus záróutasítása mind végrehajtási, mind *felírási sorrendben is* utolsó volt.

Az effektív fordítóprogramok és célprogramok konstruálásának könnyebbé tétele céljából ebben a kérdésben a FORTRAN a következő szabályokat írja elő:

1. A DO ciklus ciklusmagjához tartozik minden olyan FORTRAN utasítás, amely *felsorolási sorrendben* a DO utasítást követi és a záróutasításig található, a záróutasítást is beleértve. Ezek az utasítások mindannyian a DO ciklus, pontosabban a ciklusspecifikáció *hatáskörébe* tartoznak.

2. A FORTRAN-ban mód van a ciklusspecifikáció hatáskörének kiterjesztésére is, amikoris olyan utasítások kerülnek a ciklusmagba, melyek felsorolásuk sorrendjében nem tartoznának oda. Ehhez azonban számos speciális feltételnek kell teljesülnie. Ilyen esetben az utasítások *végrehajtási sorrendben* tartoznak a ciklusmaghoz és azt mondjuk, hogy a ciklusmag azon utasításai, melyeket az 1. pont nem sorol a ciklusspecifikáció hatáskörébe, a ciklus specifikáció ún. *kiterjesztett hatáskörébe* tartoznak.

Ilyen esetre mutat példát az alábbi programrészlet:

```
      DO 1 J=1,3
1     ISZ(J)=0
      DO 5 I=1,1000
      DO 5 J=1,3
      KD = IT(I)-IP(J)
      IF(KD) 5,2,5
2     IF(J-3) 3,25,3
3     IT(I)=IJ(J)
      ISZ(J) = ISZ(J)+1
5     CONTINUE
      .
      .
      .
25    IP(J) = 2*IP(J)-1
      GO TO 3
      .
      .
```

(3.82)

Ez a programrészlet vizsgálja az IT(I) ezer elemű tömbben levő egész számok sorozatát. Ha talál benne olyan egész számot, amely IP(1), IP(2) vagy IP(3) valamelyikével megegyezik, akkor ezen elemeket kicseréli az IJ(1), IJ(2), illetve IJ(3) elemekre. Számolja azt, hogy melyik cseréből hány történt. Abban az esetben, amikor harmadik fajta cserét hajtana végre, a 2-es címkéjű utasítás a vezérlést a ciklusspecifikáció eredeti hatáskörén kívül fekvő 25-ös címkéjű utasításra adja. Itt IP(3) értéke megváltozik, azaz a ciklus ezentúl más számokkal fogja az IT(I) számokat összehasonlítani. Innen a vezérlés visszakerül az eredeti hatáskörbe.

A 25-ös címkéjű és az őt követő utasítás tehát a ciklusspecifikáció *kiterjesztett hatáskörébe* tartozik.

A FORTRAN nyelv a következő korlátozásokat és szabályokat állítja fel a DO ciklusok kiterjesztett hatáskörével kapcsolatban:

1. Egymásba skatulyázott DO ciklusok esetén csak a legbelső DO ciklus hatásköre terjeszthető ki, és így csak ennek a hatáskörében lehet olyan vezérlésátadó utasítás, amely a kiterjesztett hatáskörben (kiterjesztésben) levő valamelyik utasításra adja a vezérlést.

2. Ez a vezérlésátadó utasítás csak olyan utasításra adhatja a vezérlést, amely nem tartozik egyetlen ciklusspecifikáció hatáskörébe sem.

3. Ugyanúgy, mint a közönséges értelemben vett hatáskör utasításai, a kiterjesztett hatáskör utasításai sem adhatnak értéket a ciklusspecifikáció változóinak.

4. A kiterjesztett hatáskör nem tartalmazhat olyan DO ciklust, amelynek újabb kiterjesztett hatásköre van.

5. A kiterjesztett hatáskör utolsó végrehajtott utasítása mindig olyan vezérlésátadó utasítás, amely a vezérlést a közönséges értelemben vett hatáskör egy utasítására adja vissza. Ennek végrehajtását mindig meg kell előznie az 1. és 2. pontokban leírt vezérlésátadásnak. Így ugyanannak a ciklusnak ugyanannyiadik lefutásába térünk vissza.

A kiterjesztett hatáskör ismeretében kissé bővebben kell magyaráznunk a DO utasítással kapcsolatos ismereteinket (lásd 69. oldal):

1. az *l* címke a közönséges értelemben vett hatáskör utolsó utasításának címkéje. Ez az utasítás mindig a hatáskörhöz és

sohasem a kiterjesztett hatáskörhöz tartozik, így felírási sorrendben mindig követi a DO utasítást.

2. A ciklus hatáskörébe kívülről beugrani csak a kiterjesztett hatáskörből szabad és csak akkor, ha ezt a beugrást megelőzte egy kiugrás a hatáskör ugyanannyiadik lefutásából.

A kiterjesztett hatáskör bevétele a FORTRAN nyelvbe történeti okokra vezethető vissza. Azt kívánták a nyelv szerzői ezzel lehetővé tenni, hogy bizonyos javításokat és betoldásokat a programozó ne legyen kénytelen teljes egészében az adott ciklusok belsejében elvégezni, hanem bizonyos esetekben ezeket a program más részén is elhelyezhesse. Konkrét gépi reprezentációk sok esetben jóval rugalmasabban kezelik ezt a kérdést. Például az ICT FORTRAN be sem vezeti a kiterjesztett hatáskör fogalmát, hanem az egész kérdéskört egyetlen szabály kimondásával tartja kézben:

„Szabad egy ciklusspecifikáció hatáskörébe beugrani, ha előzőleg ugyanezen ciklusspecifikáció hatásköréből ugrottunk ki és közben nem adtunk értéket a ciklusspecifikáció paramétereinek és a ciklusváltozónak sem."

Ha a fordítóprogram feldolgozza az olyan forrásprogramokat is, melyekben a ciklusmag értékeket ad pl. a ciklusváltozónak, és hibajelzést sem ad, akkor a fordítóprogram pontos ismeretében egy ügyes programozó igen sok programozási fogást tud az egyes gépi reprezentációk lehetőségein belül megvalósítani. Igaz, egy ilyen rendszer sok hibalehetőséget is rejt magában, és általában kompatibilitási problémákat is okoz.

6. Megállító utasítások. A PAUSE és a STOP utasítás

Befejezésül még két végrehajtható utasítással ismerkedünk meg, melyek a program futásának átmeneti, vagy végleges megállítására szolgálnak. Átmeneti megállítás érhető el a

PAUSE

vagy a (3.83)

PAUSE *n*

utasítással. Itt *n* maximálisan ötjegyű ún. *oktális szám*, azaz olyan karaktersorozat, melyben csak a 0, 1, ..., 7 számjegyek fordulhatnak elő.

Az utasítás hatására a program futása megáll, *n* hozzáférhetővé válik, például kinyomtatódik, vagy bizonyos jelzőlámpák kombinációjából leolvasható. Ezután a számológép kezelőjétől függ, hogy a programot továbbindítja-e. A tovább-

indíthatóság biztosítása szintén a PAUSE utasítás feladata. Továbbindítás esetén a PAUSE utasítást követő FORTRAN nyelvű utasítás kerül végrehajtásra.

A végleges megállító utasítások a

STOP

és a (3.84)

STOP *n*

melyek funkciója lényegesen egyszerűbb, mint a megfelelő PAUSE utasításoké: ilyenkor nincs biztosítva a továbbindíthatóság.

Tekintettel arra, hogy a programok megállítása szorosan összefügg a számológépek üzemével, ezen utasítások pontos hatása is a gépi reprezentációktól függő.

Több gépi reprezentációban az *n* számra mások a kikötések: pl. lehet tízes számrendszerbeli szám is, ugyanakkor csak utolsó két jegye hozzáférhető. A STOP utasítás végrehajtásában is szerepet kaphat az *n* szám, például kinyomtatódik. Egyes gépi reprezentációk nem ismerik a STOP utasítást, hanem helyette a „CALL EXIT" utasítást használják. (Ez szintaktikusan egy szubrutinhívó utasítás (lásd Ötödik fejezet 3*a* pontját), amely egy paraméter nélküli külső eljárást aktivizál.) Az is előfordul, hogy a STOP utasítás egyszersmind a program memóriából való törlését is előidézheti. Ezért ezeknek az utasításoknak a tulajdonságaival a program megírása előtt okvetlenül ismerkedjünk meg.

7. Gyakorlatok

1. Legyen **x** egy *N*-elemű, valós típusú vektor. Tegyük fel, hogy **x** elemei egy méréssorozat eredményei és feladatunk az átlag és az empirikus szórásnégyzet kiszámítása:

$$\bar{x} = \frac{\Sigma x_i}{N} \qquad S^2 = \frac{1}{N-1}\,\Sigma\,(x_i-\bar{x})^2$$

Írjunk programrészletet a feladat megoldására.

2. Legyen **x** és **y** is egy-egy *N*-elemű vektor, melynek összetartozó elemei egy síkbeli pont koordinátái. Illesszünk a legkisebb négyzetek módszerével egyenest a pontokra! Megköveteljük, hogy a

$$Q = \sum_{i=1}^{N} (y_i - Ax_i - B)^2$$

kifejezés az *A* és *B* együtthatók megfelelő választásával minimális értéket vegyen fel!

A feladat megoldható például a következő képletek beprogramozásával:

$$S_1 = \sum_{i=1}^{N} x_i \quad S_2 = \sum_{i=1}^{N} x^2 \quad S_3 = \sum_{i=1}^{N} y_i \quad S_4 = \sum_{i=1}^{N} x_i y_i \quad S_5 = \sum_{i=1}^{N} y_i^2$$

$$D = NS_2 - S_1^2 \qquad A = \frac{NS_4 - S_1 S_3}{D} \qquad B = \frac{S_2 S_3 - S_1 S_4}{D}$$

$$Q = S_5 - 2AS_4 + A^2 S_2 - 2BS_3 + 2ABS_1 + NB^2$$

Írjunk programrészletet a feladat megoldására!

3. Írjunk programrészletet, mely az F ezer elemű tömbbe elhelyezi az

$$f(x, y, z) = \frac{x^2 + y^2 + z^2}{x \cdot y \cdot z}$$

függvény értékeit, ha x, y és z értékei felveszik a 0.1, 0.2, ..., 1.0 értékeket minden kombinációban!

4. Tekintsük az

$$f(x_1, x_2, \ldots, x_N) = \sum_{i=1}^{N} a_i x_i \Bigg/ \sum_{i=1}^{N} b_i x_i$$

N-változós függvényt. Azt szeretnénk megvizsgálni, hogy az $(x_1^0, x_2^0, \ldots, x_N^0)$ hely környékén hogyan viselkedik a függvény értéke, ha az egyes változóknak külön-külön néhány eltérő értéket adunk, miközben a többi változó értéke x_i^0 marad. Legyen az eltérő értékek száma rendre $i_1, i_2, \ldots, i_N$, ezek maximuma legyen IMAX és az eltérő értékek legyenek a DAT nevű IMAX×N-méretű tömbben elhelyezve. Írjunk programrészletet, mely elkészíti a kívánt függvényértékeket!

5. Korábban ismertettük a Gauss—Jordan-féle algoritmust lineáris inhomogén egyenletek megoldására és alkalmaztuk mátrixok invertálására is. Egyenletmegoldásra e módszer helyett a Gauss-féle eliminációs módszert szokás alkalmazni, ami lényegében megegyezik a középiskolában is tanított „kifejezéses—behelyettesítéses" módszerrel, melynek algoritmusa a következőképpen foglalható össze:

a) I = 1, 2, ..., N−1-re végrehajtjuk a következőket:

a1) az I-edik sort az I+1-edik elemétől kezdve végigosztjuk a_{II}-vel;

a2) az I-edik sor alatti J-edik sorok (J = I+1, I+2, ..., N) K-adik (K = I+1, I+2, ..., N+1) elemeiből rendre levonjuk az I-edik sor K-adik elemének a_{JI}-szeresét és az eredményt visszahelyezzük a J-edik sor K-adik elemébe;

b) az egyenletrendszer N-edik ismeretlene ezután közvetlenül kiszámítható:

$$x_{\mathrm{N}} = a_{\mathrm{N\,N+1}}/a_{\mathrm{NN}}$$

c) a további ismeretleneket a visszahelyettesítés során kaphatjuk meg, melynek algoritmusa a következő:

$I = N-1, N-2, \ldots, 2, 1$-re rendre kell képezni a mátrix megváltoztatott elemeivel az

$$S_I = a_{I\,I+1} x_{I+1} + a_{I\,I+2} x_{I+2} + \ldots + a_{I\,N} x_N$$

szorzatösszegeket és le kell azokat vonni $a_{I\,N+1}$-ből.

Az algoritmus működésének feltételei ugyanazok, mint a Gauss—Jordan-eljárásé.

Írjunk programrészletet, mely realizálja az adott algoritmust!

6. Legyen **A** $N \times N$-es mátrix, **x** és **b** N-elemű oszlopvektor. A felhasználásukkal felírható

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{3.85}$$

x-re vonatkozó inhomogén lineáris egyenletrendszer megoldására a gyakorlati esetekben iteratív módszerek is felhasználhatók, ha például az **A** mátrix minden egyes sorára fennáll

$$|a_{II}| > \sum_{J=1}^{I-1} |a_{IJ}| + \sum_{J=I+1}^{N} |a_{IJ}| \tag{3.86}$$

avagy az **A** mátrix az egyenletek sorrendjének felcserélésével ilyen alakra hozható. Ebben az esetben a (3.85) egyenletet az iterációra alkalmas

$$\mathbf{x} = \mathbf{P} \cdot \mathbf{x} + \mathbf{b}_1 \tag{3.87}$$

alakra lehet hozni, ahol a **P** mátrix minden egyes sorára teljesül

$$\sum_{J=1}^{N} |p_{IJ}| < 1 \tag{3.88}$$

(3.87)-ben a jobb oldalba **x** k-adik közelítését helyettesítve, a bal oldalon **x** $(k+1)$-edik közelítése adódik.

Feltételezve, hogy az **x** vektor a megoldás egy közelítését tartalmazza, írjunk FORTRAN programot (3.87) iteratív megoldására a vázolt iteráció felhasználásával!

7. Tegyük fel, hogy fennállnak (3.85)-re a (3.86) feltételek. Fejezzük ki a I-edik sorból az I-edik ismeretlent:

$$x_I = \left(b_I - \sum_{K=1}^{I-1} a_{IK} x_K - \sum_{K=I+1}^{N} a_{IK} x_K \right) \Big/ a_{II} \qquad I = 1, 2, \ldots, N \tag{3.89}$$

Ezek az egyenletek éppen úgy alkalmasak iterációs megoldásra, mint a (3.87) egyenletrendszer, sőt nem kívánják meg az $\mathbf{A} \to \mathbf{P}$ mátrix-előállítást sem. Írjunk programrészletet, mely (3.89) szerint közvetlenül a (3.85) egyenletrendszert oldja meg iterációval, a (3.87) alak megkerülésével!

8. Az előző két gyakorlat megoldásaiban együtt kellett „tárolni" a k-adik és a $(k+1)$-edik megoldást. Sok esetben a feladat méretei ezt nem teszik lehetővé. A Gauss—Seidel-féle iterációs eljárás ezért úgy jár

el, hogy minden egyes komponens kiértékeléséhez az összes többi komponens „legfrissebb" értékét használja fel. Kimutatható, hogy a konvergencia ezáltal csak javul, sőt a program is egyszerűsödik! Írjuk meg ezt az egyszerűbb programrészletet!

9. A Gauss—Seidel-féle iterációs algoritmus soronként egy x komponens értékét javítja meg. A javítás

$$\Delta x_{\mathsf{I}} = x_{\mathsf{I}}^{k+1} - x_{\mathsf{I}}^{k}.$$

Az ún. overrelaxációs módszerek abban térnek el a Gauss—Seidel-féle iterációtól, hogy ezt a javítást egy ω konstans szorzófaktorral megnövelve veszik figyelembe. Természetesen ω megválasztása részletesebb vizsgálatokat igényel. A konvergencia sebessége ily módon erősen megnövelhető. Olykor ω időnkénti megváltoztatását is beépítik a programba. Írjunk olyan programrészletet (3.85) megoldására, mely overrelaxációs módszerrel dolgozik!

Megjegyezzük, hogy a 8-as gyakorlat a 9-esnek speciális esete, mégpedig amelyben $\omega = 1$.

10. Legyen A szimmetrikus, N×N-es mátrix. Tekintsük az

$$\mathbf{Ax} = \lambda\mathbf{x} \tag{3.90}$$

ún. *sajátértékegyenletet*, melyben λ az A mátrix ún. *sajátértéke*, $\mathbf{x}$ pedig ún. *sajátvektora*. A sajátértékek maximális száma éppen N. Minden sajátértékhez legalább egy sajátvektor tartozik.

Adott A szimmetrikus mátrix esetén a legnagyobb abszolút értékű sajátérték és a hozzá tartozó sajátvektor meghatározására a következő iterációs eljárás alkalmas:

a) Választunk egy $\mathbf{x}^{(0)}$ kezdővektort, melynek maximális abszolút értékű komponense 1-gyel egyenlő, valamint egy $\lambda^{(0)}$ számot. A sajátvektor nulladik közelítése $\mathbf{x}^{(0)}$, a sajátértéké pedig $\lambda^{(0)}$.

b) Képezzük az $\mathbf{y}^{(k)} = \mathbf{Ax}^{(k)}$ segédvektort, és megvizsgáljuk, hogy fennáll-e minden J-re valamely adott EPS-nél nem nagyobb hibával az

$$\mathbf{y}_J^{(k)} = \lambda^{(k)} \mathbf{x}_J^{(k)} \tag{3.91}$$

egyenlőség. Ha ez az egyenlőség minden J-re teljesül, akkor áttérünk a *d)* lépésre, ellenkező esetben a *c)* lépésre.

c) Legyen az $\mathbf{y}^{(k)}$ vektor maximális abszolút értékű eleme $y_{\mathsf{MAX}}^{(k)}$, és jelentse MAX ennek indexét. Képezzük a

$$\lambda^{(k+1)} = y_{\mathsf{MAX}}^{(k)} / x_{\mathsf{MAX}}^{(k)}$$

és $$\tag{3.92}$$

$$\mathbf{x}^{(k+1)} = \mathbf{y}^{(k)} / \lambda^{(k+1)}$$

újabb becsléseket, majd térjünk vissza a következő iterációs lépés elvégzése céljából a *b)* pontra.

d) A keresett sajátérték a $\lambda^{(k)}$ szám, a keresett sajátvektor pedig az

$$\mathbf{x}^{(k+1)} = \frac{\mathbf{y}^{(k)}}{|\mathbf{y}^{(k)}|}$$

vektor ($\mathbf{x}^{(k+1)}$ abszolút értéke 1-gyel egyenlő).

Készítsünk programrészletet a vázolt iterációra!

11. A (3.90) sajátértékegyenlet λ sajátértékei gyökei egy ún. *karakterisztikus polinomnak* is:

$$p(\lambda) = \lambda^N + \alpha_1\lambda^{N-1} + \alpha_2\lambda^{N-2} + \dots \quad \dots + \alpha_{N-1}\lambda + \alpha_N \tag{3.93}$$

A karakterisztikus polinom α_I együtthatóinak meghatározására használható a Le Verrier-féle algoritmus. Legyenek $\mathbf{A}, \mathbf{U}^I, \mathbf{V}^I$ N×N-es mátrixok és használjuk a Spur $(\mathbf{A}) = \sum_{\mathsf{I}=1}^{\mathsf{N}} a_{\mathsf{II}}$ jelölést; ekkor az algoritmus a következőképpen adható meg:

a) $\mathbf{V}^{(0)} = \mathbf{E}$, az N×N-es egységmatrix

b) Minden I-re, I = 1, 2, ..., N végezzük el a következőket:

b1) $\mathbf{U}^{(\mathsf{I})} = \mathbf{A}\cdot\mathbf{V}^{(\mathsf{I}-1)}$

b2) $\alpha_\mathsf{I} = -\dfrac{1}{\mathsf{I}}\cdot\mathrm{Spur}(\mathbf{U}^{(\mathsf{I})})$

b3) $\mathbf{V}^{(\mathsf{I})} = \mathbf{U}^{(\mathsf{I})} + \alpha_\mathsf{I}\cdot\mathbf{E}$

Így képződött az összes α_I együttható. Kimutatható, hogy $\mathbf{V}^{(\mathsf{N})}$ értéke a $\mathbf{0}$ nullmátrix. Ezt a tényt felhasználhatjuk a pontosság ellenőrzésére.

Írjunk programrészletet a feladat megoldására!

Negyedik fejezet

DEKLARATÍV UTASÍTÁSOK (NEMVÉGREHAJTHATÓ UTASÍTÁSOK)

1. A FORTRAN program szerkezete. A szegmensek

A Harmadik fejezetben megismerkedtünk szinte valamennyi végrehajtható utasítással, csupán az input-output utasítások és a CALL, vagyis az eljáráshívó utasítás tárgyalása maradt hátra (ezeket a Hatodik fejezetben, illetve az Ötödik fejezet 3. pontjában ismertetjük).

A megismert utasítások segítségével gyakorlatilag minden szóbajövő problémára tudunk megfelelő programrészletet készíteni. De — mint már korábban jeleztük — ahhoz, hogy ezeket a részleteket a fordítóprogram megértse, további információkra van szüksége, amelyeket az ún *deklaratív*, más néven *nemvégrehajtható* utasításokban adhatunk meg. Ha a végrehajtható és nemvégrehajtható utasításokat meghatározott helyes sorrendben fűzzük össze, a fordítóprogram el fogja készíteni a feladat megoldásához szükséges célprogramot.

A programozói gyakorlat igen érdekes szempontját veszi figyelembe a FORTRAN nyelv, amikor lehetővé teszi nemcsak teljes, működőképes programoknak, hanem a programok bizonyos önállósággal rendelkező részeinek, az ún. *programegységeknek*, vagy *szegmenseknek* külön-külön lefordítását. Így új programok esetleg gyorsan összeállíthatók kész, hibátlan szegmensekből, vagy pedig kidolgozásukon több programozó viszonylag függetlenül dolgozhat. A *szegmensenkénti fordíthatóság elve* természetesen megköveteli, hogy a különféle típusú utasítások helyes sorrendjét már a szegmenseken belül biztosítsuk. A szegmensek sorrendje és összefűzése, illetve a szegmensek együttműködésének biztosítása külön feladat, melyet az Ötödik fejezetben tárgyalunk.

Az ALGOL nyelvű programozásban jártas olvasó számára megjegyezzük, hogy a FORTRAN programok szegmensei nagyjából az ALGOL blokkok és eljárások megfelelőinek tekinthetők. Ezek a szegmensek azonban mellérendeltségi (és nem alárendeltségi) viszonyban vannak egymással.

Jelen fejezetünknek az a célja, hogy áttekintsük az összes nemvégrehajtható utasítást és elhelyezésük szabályait. Részletesen tárgyaljuk a nemvégrehajtható utasítások közül azokat, amelyek egyetlen szegmensből álló programok írásához szükségesek. Az input-output utasítások vezérléséhez szükséges FORMAT utasítást a Hatodik fejezetben tárgyaljuk részletesen. Megállapításainkat szegmensekre fogjuk kimondani, hogy értelemszerűen átvihetők legyenek több szegmensből álló programokra is. Ez alól egyetlen kivételt kell tennünk: az egy szegmensből álló program leírását vagy listáját mindig a MASTER utasítással fogjuk kezdeni és az END utasítással fogjuk befejezni (lásd az Ötödik fejezet 2. pontját). A MASTER utasítás alakja

MASTER *sz* (4.1)

ahol *sz* tetszőleges azonosító (a szegmens neve). Az END utasítás az END betűk leírásából áll. Egyik utasításnak sem lehet címkéje. (A MASTER utasítás nem része az ASA FORTRAN nyelvnek.)

A FORTRAN programok tehát szegmensekből, a szegmensek pedig utasításokból állnak. A szegmenseken belül megfelelő sorrendben kell elhelyezni a végrehajtható és nemvégrehajtható utasításokat.

2. A nemvégrehajtható utasítások fajtái. Az utasítások elhelyezése a szegmenseken belül

A FORTRAN nyelv a következő nemvégrehajtható utasításokat ismeri:

1. *Szegmensnyitó utasítások:* a MASTER, SUBROUTINE, FUNCTION és BLOCK DATA utasítások, lásd az Ötödik fejezet 2., ill. 6. pontját.

2. *Specifikáló utasítások:*

a) Típusdeklaráló utasítások: az INTEGER, REAL, DOUBLE PRECISION, COMPLEX és LOGICAL utasítások, lásd a Negyedik fejezet 3*a)* alpontot.

b) Tömbdeklaráció (DIMENSION) *utasítás,* lásd a 3*b)* alpontot.

c) Közös adatmező deklaráció (COMMON) *utasítás,* lásd az Ötödik fejezet 5. és 6. pontját.

d) Az EQUIVALENCE utasítás, lásd a 4. pontot és részben az Ötödik fejezet 5. pontját.

e) Az EXTERNAL utasítás, lásd az Ötödik fejezet 7. pontját.

3. *Kezdőértékadó* (DATA) *utasítás*, lásd az 5. pontot és részben az Ötödik fejezet 6. pontját.

4. A FORMAT utasítás, lásd főként a Hatodik fejezet 4. pontját.

5. *Utasítás függvény*, lásd a 6. pontot.

6. *Szegmenszáró* (END) *utasítás*, lásd az Ötödik fejezet 2. pontját.

Ezen nemvégrehajtható utasítások és a végrehajtható utasítások sorrendje nem lehet tetszőleges egy szegmensen belül. Bár erre vonatkozóan az [1] nyelvdefiníció nem ad meg minden tudnivalót, a fontosabb gépi reprezentánsok ismeretében a következő két szabályt állíthatjuk fel:

1. Szegmenseken belül kötelező az utasítások között a következő sorrendet betartani:

a) szegmensnyitó utasítás;
b) specifikáló és kezdőértékadó utasítások, egymás közt tetszőleges sorrendben;
c) utasítás függvények;
d) végrehajtható utasítások;
e) szegmenszáró utasítás.

A *b)*-t és *c)*-t tartalmazó szegmensrészt együtt *deklarációs résznek* szokás nevezni.

2. FORMAT utasításokat bárhol el lehet helyezni a szegmens belsejében, azaz a szegmensnyitó és szegmenszáró utasítások között.

A fordítóprogramok a specifikáló és kezdőértékadó utasításokat általában meghatározott sorrendben dolgozzák fel: típusdeklarációk, tömbdeklarációk, COMMON utasítások, EQUIVALENCE és DATA utasítások. Abból a célból, hogy ne kelljen a fordítóprogramnak ezeket az utasításokat külön sorbaszedni, az egyes gépi reprezentánsok ezen utasítások között is kötelező megadási sorrendeket írhatnak elő. Megjegyezzük továbbá, hogy bár a FORTRAN nyelv definíciója nem zárja ki — a FORMAT utasítások mintájára — a DATA utasítások tetszőleges elhelyezését sem, de ez a legtöbb gépi reprezentációban tiltva van.

A sorrendiségen kívül itt említjük meg a nemvégrehajtható utasítások néhány további általános tulajdonságát:

1. *A nemvégrehajtható utasításoknak nem lehet* olyan *címkéje,* melyre vezérlésátadó utasítás hivatkozik. *A* FORMAT *utasításnak mindig van címkéje,* erre azonban az input-output utasítások hivatkoznak.

2. A specifikáló és a kezdőértékadó (DATA) utasítások azonos szerkezetűek:

$$\mathrm{U}\, l_1, l_2, \ldots, l_N \tag{4.2}$$

ahol U egy FORTRAN alapszót jelöl, mely megadja az utasítás fajtáját (sokszor elnevezését is), l_1, l_2, ..., l_N pedig egy N-elemű *lista,* amelyben az l_I, $I=1, 2, \ldots, N$ listaelemeket vesszők választják el egymástól. A lista állhat egyetlenegy listaelemből is. A listaelemek szerkezetét az egyes U alapszavak szabják meg, amint azt az egyes utasítások kapcsán részletesen tárgyalni fogjuk.

3. Ha több azonos fajtájú specifikáló vagy kezdőértékadó utasítás van egy szegmensben, akkor ezek hatásuk megváltozása nélkül egyesíthetőek úgy, hogy az alapszót csak egyszer írjuk ki és a megfelelő listákat eredeti sorrendjükben vesszővel elválasztva egymás mögé írjuk. Például az

```
INTEGER A, B, C
```

és az (4.3)

```
INTEGER A1, C1, DADA
```

utasítások egyesítése:

```
INTEGER A,B,C,A1,C1,DADA
```
(4.4)

4. Előző megállapításunk fordítottja is igaz: szabad egyetlen specifikáló vagy kezdőértékadó utasítást több utasítássá szétbontani. Erre pl. ok lehet az, hogy az utasítás túl hosszú: nem fér ki 20 sorban.

Ezek előrebocsátása után áttérhetünk az egyes nemvégrehajtható utasítások részletes tárgyalására.

3. Típus- és tömbdeklarációk

a) Típusdeklarációk. Az INTEGER, REAL, DOUBLE PRECISION, COMPLEX és LOGICAL utasítás

Eddig az értékek és változók típusáról beszéltünk, megemlítve azt is, hogy a tömböknek, sőt a függvényértékeknek is van típusa. Ez azt jelenti, hogy ha egy szegmens belsejében található azonosító valamilyen változót, tömböt vagy függvényt jelöl, akkor egyben kifejezi az általa képviselt mennyiség típusát is; pl. I önmagában lehet az egész típusú változó azonosítója, az IABS azonosító egész típusú standard függvény azonosítója (lásd a 4. táblázatot!) stb.

Ismerjük a FORTRAN nyelv automatikus típusdeklarációs szabályát (l. Második fejezet 3. pontját), mely szerint az I, J, K, L, M és N betűvel kezdődő azonosítók egész típusú mennyiségeket jelölnek, a többi betűvel kezdődőek pedig valós típusúakat, hacsak ezzel ellentétes információt nem tartalmaz a szegmens. E szabály biztosítja INTEGER és REAL típusú mennyiségek típusának automatikus meghatározását. Sokszor előnyös lehet azonban, ha egy bizonyos azonosító típusát e szabálytól eltérő módon is előírhatjuk. DOUBLE PRECISION, COMPLEX és LOGICAL típusú mennyiségekkel pedig amúgy is csak akkor tudunk számolni, ha előzőleg deklaráljuk őket. A *típusdeklaráló utasítások* segítségével mindez keresztülvihető. Az utasítás alakja (4.2) mintájára.

$$\mathrm{T}\ l_1, l_2, \dots \dots, l_N \tag{4.5}$$

ahol T helyén a következő alapszavak valamelyike áll:

INTEGER

REAL

DOUBLE PRECISION

COMPLEX

LOGICAL

Az l_i listaelemek pedig azon mennyiségek azonosítói, melyek típusát elő akartuk írni: változóké, tömböké vagy függvényeké, vagy pedig esetleg tömbdeklarátorok (lásd a *b)* és *c)* alpontokat). A deklaráció a lista minden elemére vonatkozik, függet-

lenül attól, hogy ez esetleg ellentmond-e az automatikus típusdeklarációs szabálynak.

A deklarációk tehát a listaelemekre vonatkozóan az *automatikus típusdeklarációs szabályt hatályon kívül helyezik.*

A típusdeklarációk általában a szegmens deklarációs részének elején helyezkednek el. Példák:

INTEGER A,A1,F2,M,Q7 (4.6)

DOUBLE PRECISION W,DW,Q (4.7)

LOGICAL F1,F3,C9 (4.8)

REAL M1,M2,M3 (4.9)

Ügyeljünk arra, hogy *ugyanaz az azonosító egyetlen szegmensen belül nem szerepelhet több típusdeklaráló utasításban.*

b) Tömbdeklarációk. A DIMENSION utasítás

A Harmadik fejezet 4. pontjában foglalkoztunk az indexes változók használatával és megállapítottuk, hogy

1. az indexes változó tömbelem;
2. a tömböt mindig deklarálni kell;
3. az indexes változó aktuális indexei mindig pozitív számok, melyek nem haladhatják meg a deklarációban megadott megfelelő maximális indexértékeket;
4. a FORTRAN nyelvben maximálisan háromindexű tömböket használhatunk.

Bár, mint a *c)* alpontban látni fogjuk, tömbök deklarációja más utasításokkal is elvégezhető, speciálisan tömbök deklarálására való a DIMENSION utasítás. Például a

DIMENSION A(8), B(12,3,4), C(5,5) (4.10)

utasítás három tömböt deklarál: az A nyolcelemű vektort; a B háromdimenziós tömböt, melynek indexei rendre maximálisan a 12, 3 és 4 értékeket vehetik fel; végül a C 5×5-ös mátrixot.

Ez a felírásmód (mely a típusdeklaráló és a DIMENSION utasításon kívül a COMMON utasításban is felléphet) speciális jelentésű: a zárójel tartalma megadja az indexek számát, valamint azt is, hogy az egyes indexek értéke $+1$-től kezdve milyen maximális értéket vehet fel, azaz megadja a tömb méreteit. Az

így felírt zárójeles szerkezeteket, mint pl. A(8) vagy B(12, 3, 4), *tömbdeklarátoroknak* nevezzük. (Ha a tömb egy szegmensből álló programban van deklarálva, a tömbdeklarátorban szereplő maximális indexek csakis pozitív egész számok lehetnek. Az Ötödik fejezet 4. pontjában látni fogjuk, hogy mikor használható egész szám helyett egész típusú változó azonosítója.)

A (4.10) utasítás alakja tehát megfelel a (4.2) sémának: itt U a DIMENSION alapszó, a listaelemek pedig a tömbdeklarátorok. További példák:

```
DIMENSION ITER(12,3), LOG(25,4)          (4.11)
DIMENSION LA(4,2500), B(2,2,2)           (4.12)
```

c) A típus- és tömbdeklarációk kölcsönhatása

Tekintsük a következő programrészletet:

```
INTEGER F, F1, F2, U
REAL M1, M2, IZA, GOOL                          (4.13)
DIMENSION F(15), U(7,2), IZA(8,6), B(12)
```

Az előző alpontok szerint könnyen értelmezhető e részlet: bizonyos azonosítók típusát az első két utasítás adja meg; a GOOL azonosító felsorolása a középső utasításban felesleges (mivel az automatikus típusdeklarációval összhangban van), de nem hibás; a harmadik, a DIMENSION utasításban olyan tömbök is deklarálva vannak, amelyeknek a típusát az előző utasítások adják meg.

Az *a)* alpontban megemlítettük azt, hogy típusdeklaráló utasításban is előfordulhat tömbdeklarátor. Ennek segítségével a (4.13) részlet a következő egyenértékű formában is felírható:

```
INTEGER F(15), F1, F2, U(7,2)
                                        (4.14)
REAL M1, M2, IZA(8,6), GOOL, B(12)
```

A FORTRAN nyelvben tehát tömbök deklarására nem csak egy lehetőség van. Bemutatunk még néhány példát:

```
INTEGER ELEMEK (12,4), B, INONE
                                        (4.15)
DIMENSION B(2), D(13,6)
```

```
LOGICAL F1, F2, F3, FG(12,12), CIO
DIMENSION CIO(13), FLUX(25,10,18)                    (4.16)
```

```
DOUBLE PRECISION MATR(15,15), W, K
COMPLEX CO1, CO2                                     (4.17)
DIMENSION CO2(50), K(15), B(6,6)
```

A részletek elemzését az olvasóra bízzuk.

Felhívjuk a figyelmet arra, hogy a tömbméreteket a COMMON utasítás (lásd az Ötödik fejezet 5. pontját) segítségével is meg lehet adni. Azonban mindenképpen be kell tartani a következő szabályokat:

1. A DIMENSION utasításban tömbdeklarátorok vannak felsorolva; magukban álló tömbazonosítók itt nem fordulhatnak elő.

2. Ha egy tömbazonosító több deklaratív utasításban — típus, DIMENSION, COMMON — fordul elő, csak az egyik utasításban szerepelhet tömbdeklarátor alkotórészeként.

Példaként a 7. táblázatban megadjuk néhány tömb adatait:

7. táblázat

A tömb					
azonosítója	mérete	típusa	azonosítója	mérete	típusa
AZOR	1000	R	OZORA	42×3	L
ALI	10×10	I	OGIL	20000	R
BEA	5×5	L	PACC	2	I
B2	3×8×2	DP	Q1	4×3	L
CECIL	2×2	C	Q2	4×2	L
CICA	4×5	I	RHO	20	I
CILI	3×3	R	SL	12×3	DP
DORA	8×2	L	TATA	60×1000	R
EVA	6×6×6	I	UZ	14×14	C
JENO	12×3	R	WYL	100×4	I
L13T	32	DP	XXI	25×25	R
NEON	20	R	YBL	3×4×8	L
N25	3×3×3	DP	ZOO	75×2	L

Kérjük kedves olvasónkat, hogy gyakorlásképpen lehetőleg többféle olyan deklarációs részt írjon, mely e tömböket deklarálja. Egy megoldást példaképpen bemutatunk:

```
      INTEGER ALI(10,10), CICA(4,5), EVA(6,6,6),
     1PACC(2), RHO(20), WYL(100,4)
      REAL AZOR(1000), CILI(3,3), JENO(12,3), NEON(20),
     1XXI(25,25)
      DOUBLE PRECISION B2, L13T, N25(3,3,3), SL(12,3)
      LOGICAL BEA, DORA, YBL, ZOO, Q1(4,3), Q2(4,2),
     1OZORA
      COMPLEX CECIL(2,2), UZ(14,14)
      DIMENSION BEA(5,5), B2(3,8,2), DORA(8,2), L13T(32),
     1OZORA(42,3), OGIL(20000), TATA(6,1000), YBL(3,4,8),
     2ZOO(75,2)
```

d) A tömbök elhelyezkedése a számológép memóriájában. A tömbelem sorszáma és címe

A FORTRAN programozási nyelv néhány szabályát sokkal könnyebb megfogalmazni és megérteni, ha azokat a számológép operatív memóriájához kapcsoljuk. A memória *szavakra* vagy *rekeszekre* van osztva. Egy számadat elhelyezéséhez egy memóriarekesz szükséges. (Komplex és kétszeres pontosságú változók esetén kettő, de ettől pillanatnyilag a könnyebb érthetőség kedvéért tekintsünk el.)

A számológép memóriáját (gyorsmemóriáját) úgy képzeljük el, mint memóriarekeszek lineáris sorozatát. Egy ilyen sorozat elemeit meg lehet számozni: a 2. ábrán egy memóriarészletet szemléltetünk, a 418-as rekesztől a 449-es rekeszig. A rekeszek sorszámát a számológépes gyakorlatban *címeknek* hívják. Például azt a tényt, hogy a J változónak a memóriában a 429-es rekesz felel meg, úgy is mondhatjuk, hogy „a 429-es címen

van elhelyezve", vagy „a 429-es címen van", vagy egyszerűen: „J címe 429".

Aki FORTRAN nyelven programoz, annak általában nem kell törődnie a címek pontos számértékével. Viszont érdekelheti az, hogy adott program esetén mi van a memóriában a J változó rekesze előtt és mi utána. Különösen fontos részletesen megtárgyalnunk az indexes változók memóriában való elhelyezésének a kérdését.

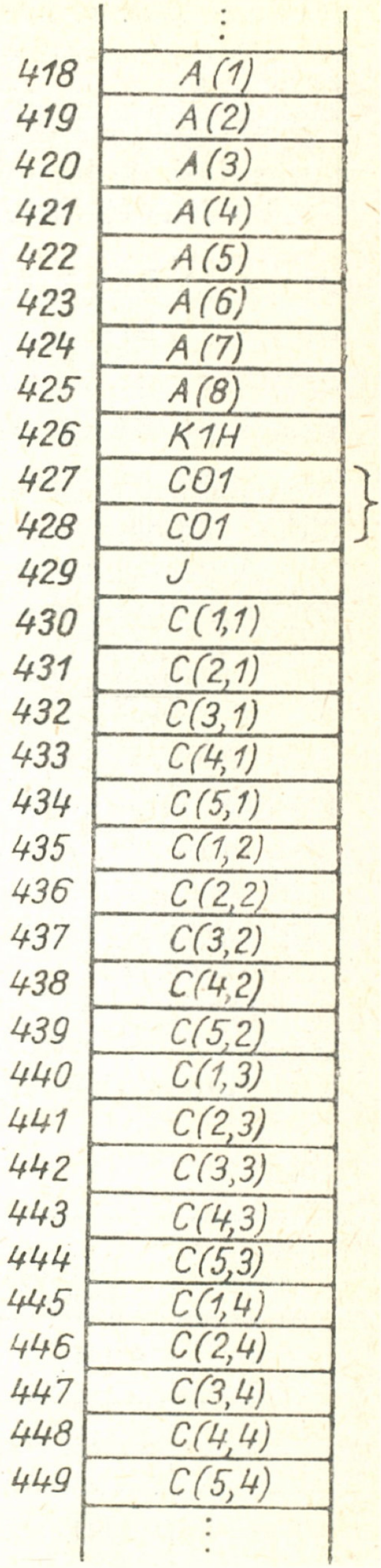
A 2. ábra memóriarészlete az A tömb elemeivel kezdődik. A 8 elem növekvő indexek szerint követi egymást. Ez általában is így van: a tömbelemek mindig valamilyen *sorszám* szerint vannak rendezve, mely az egydimenziós tömbök esetén megegyezik az index értékével. Például A(5) sorszáma 5.

Bár egy tömbelem tényleges címe — pl. az A(5) tömbelem esetén a 422 — érdektelen, sokszor szükség lehet arra, hogy a tömb első elemének — itt A(1)-nek — címével való kapcsolatát ismerjük. Jelöljük A(1) címét A-val. Akkor az A(I) tömbelem k(I) címe a

$$k(\mathrm{I}) = A + (\mathrm{I} - 1) \qquad (4.18)$$

összefüggésből számítható. A (4.18) összefüggés jobb oldalán álló kifejezést *címfüggvénynek* fogjuk nevezni, melyben az A-hoz járuló additív tag az A(I) tömbelem A-ra vonatkozó *relatív címe.*

Valamivel összetettebb a helyzet a több indexű, vagyis többdimenziós tömbök esetén. Ezeket előbb valamilyen



2. ábra

rendszer szerint lineáris sorba kell rendezni. A 2. ábrán azt a rendszert mutatjuk be a C 5×4-es tömb kapcsán, amit a FORTRAN nyelv szabályai a kétdimenziós tömbökre előírnak. Ezt *oszlopfolytonos elrendezésnek* nevezzük, és könnyen általánosíthatjuk háromdimenziós tömbök esetére is. Például a B 4×2×3-as tömb elemei a következő sorba rendezhetők:

B(1, 1, 1), B(2, 1, 1), B(3, 1, 1), B(4, 1, 1), B(1, 2, 1), B(2, 2, 1), B(3, 2, 1), B(4, 2, 1), B(1, 1, 2), B(2, 1, 2), B(3, 1, 2), B(4, 1, 2), B(1, 2, 2), B(2, 2, 2), B(3, 2, 2), B(4, 2, 2), B(1, 1, 3), B(2, 1, 3), B(3, 1, 3), B(4, 1, 3), B(1, 2, 3,), B(2, 2, 3), B(3, 2, 3), és B(4, 2, 3).

A sorozaton belül elemről elemre haladva leggyorsabban az első index, leglassabban az utolsó index növekszik.

A két- és háromdimenziós tömbök elemeinek sorszámát és címfüggvényüket a 8. táblázatban adjuk meg. Ezek a kifejezések már tartalmazzák bizonyos indexek maximális értékét is: ezért szerepel a táblázatban a tömbdeklarátor.

A FORTRAN programokban egy COMPLEX változó kétszer annyi memóriarekeszt foglal, mint egy REAL változó. Kérdés, hogy melyik rekesz címét tekintsük a COMPLEX változó címének. Általános érvényű szabály szerint valamely *változó címének* mindig *a hozzárendelt legelső memóriarekesz címét* kell tekinteni. Például a 2. ábra CO1 COMPLEX típusú változójának címe 427.

Ha egy tömbelem tárolásához több memóriarekesz szükséges, ez befolyásolja a címfüggvény szerkezetét is. A 8. táblázat utolsó képletei ekkor módosulnak: a kapcsos zárójelek

Tömbelemek sorszáma

Dimenziók száma	A tömbök deklarátora	A maximális sorszám	Az aktuális tömbelem	Az első elem címe
1	A(N1)	N1	A(I)	*A*
2	B(N1, N2)	N1 * N2	B(I, J)	*B*
3	C(N1, N2, N3)	N1 * N2 * N3	C(I, J, K)	*C*

mögé egy *d* szorzófaktor kerül, ahol *d* azon memóriarekeszek száma, melyek egy tömbelem tárolásához szükségesek.

Egyes gépi reprezentációkban *d* értéke REAL tömbök esetén is különbözhet 1-től. Azonban az ASA FORTRAN szabályai előírják, hogy a COMPLEX és DOUBLE PRECISION típusú mennyiségekre vonatkozó *d* értéke pontosan kétszerese legyen a REAL, INTEGER és LOGICAL típusú változókra vonatkozó *d* értéknek.

4. Az EQUIVALENCE utasítás

A számológépek egyik drága és korlátlanul nem növelhető része a gyorsmemória. Bizonyos problémák megoldásához pedig igen nagy memóriakapacitásra lenne szükség. A programozó ezért már a program írása közben gondol arra, hogyan használja ki a memóriakapacitást. Az EQUIVALENCE utasítás ebben nagy segítségére lehet: lehetővé teszi, hogy a program végrehajtása során, időben egymás után a memóriában ugyanazon a címen több változót helyezhessen el; vagy hogy ugyanazt a mennyiséget egyidőben több néven is nevezhesse. Utóbbira például akkor kerülhet sor, ha ugyanazon szegmens különböző részeit különböző személyek programozzák.

Az EQUIVALENCE utasítás szintén specifikáló utasítás, tehát formailag (4.2) alakú: az EQUIVALENCE alapszót *ekvivalenciacsoportokból* álló lista követi, vesszőkkel elválasztva. Egy ilyen csoport:

$$(a_1, a_2, \ldots, a_M)$$

formailag ismét lista, zárójelpárba zárva. Itt az *elemek száma*

és a címfüggvények *8. táblázat*

Az aktuális tömbelem	
sorszáma	címe
I	$A+\{I-1\}$
I+N1*(J−1)	$B+\{I-1+N1*(J-1)\}$
I+N1*(J−1)+N1*N2*(K−1)	$C+\{I-1+N1*(J-1)+$ $+N1*N2*(K-1)\}$

legalább kettő. Az *ekvivalenciacsoport elemei változók*, vagy *konstans indexű tömbelemek* lehetnek:

EQUIVALENCE (A, B), (D(1), E(3)), (Q, P(11, 2)) (4.19)

EQUIVALENCE (C(2), F(5)), (C(6), G(6), H(1, 7)) (4.20)

A (4.19) utasítás 3 ekvivalenciacsoportot tartalmaz, mindegyik csoportban 2—2 listaelemmel. Ezek változók vagy tömbelemek, egymással tetszőlegesen kombinálva. A (4.20) utasításban egy háromelemű ekvivalenciacsoport is van.

Az EQUIVALENCE utasítás azt írja elő, hogy a programban az egyes ekvivalenciacsoportok összes elemének csoportonként azonos címe legyen. Például (4.19) hatására az A és B változók címe azonos, ugyanígy a D(1) és az E(3) tömbelemek címe megegyezik, végül a Q változó és a P(11, 2) tömbelem címe is azonos. A (4.20) utasítás hatására a C(2) és F(5) tömbelemek címei azonosak, majd a második csoportnak megfelelően C(6), G(6) és H(1,7) címe egyezik meg.

Szorítkozzunk tárgyalásunkban arra az esetre, amikor egy-egy ekvivalenciacsoporton belül az egyes elemek típusa azonos, például REAL. Ebben az esetben majdnem mindegy, hogy egy bizonyos végrehajtható utasításba a csoport melyik elemét írom be. Ha egy szegmens elején fel volt írva a (4.19) utasítás, a szegmens következő részletei ugyanarra az eredményre vezetnek:

```
      DET = D(1) - A
      IF(DET) 1,2,3
    1 C = ABS(E(3))
    2 C = C + Q
    3 CONTINUE
```

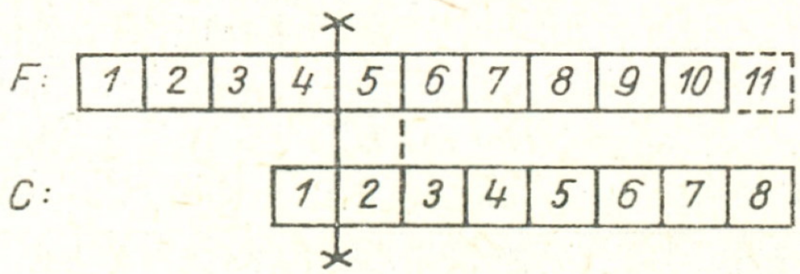
```
      DET = E(3) - A
      IF(DET) 1,2,3
    1 C = ABS(D(1))
    2 C = C + P(11,2)
    3 CONTINUE
```

```
      DET = E(3)-B
      IF(DET) 1,2,3
1     C = ABS(E(3))
2     C = C+Q
3     CONTINUE
```

A számítás időszükséglete természetesen nem azonos, hiszen indexes változókkal mindig lassúbb a számolás.

A (4.20) utasítás ekvivalenciacsoportjaiban tömbelemek vannak. Ebben az esetben az EQUIVALENCE utasítás különösen érdekes tulajdonságokat mutat. Az előző pont *d)* alpontjában láttuk, hogy a tömbelemek meghatározott sorrendben helyezkednek el a memóriában. Ez fennáll például a (4.20) utasítás első ekvivalenciacsoportjában előfordult C és F tömbökre is. De ha C(2) és F(5) azonos címen helyezkedik el, akkor ennek alapján azonos címeken kell elhelyezkednie a C(1), F(4), a C(3), F(6), a C(4), F(7) stb. elempároknak is. Ebből következik, hogy ha tömbelemek szerepelnek ekvivalenciacsoportban, akkor az ilyen csoport ekvivalenciába hoz további elemeket is a tömbelemek címe szerint.

Legyenek a (4.20) utasításban szereplő tömbök deklarátorai, melyek tehát a maximális indexeket tartalmazzák, C(8), F(10), G(10) és H(2,9). Az első ekvivalenciacsoport hatását a 3. ábrán láthatjuk. Az ekvivalenciacsoport nyújtotta összekapcsolódást a $\overset{\times}{\underset{\times}{|}}\,\vdots$ jellel szemléltetjük.

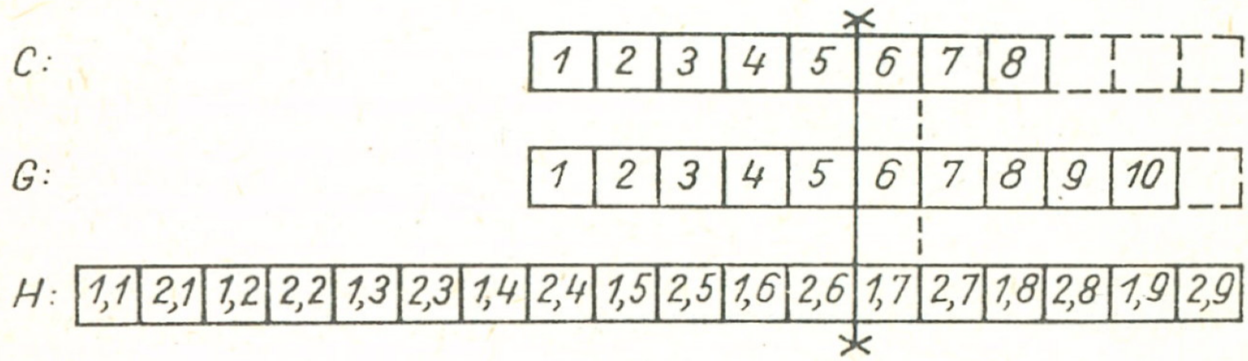


3. ábra

Tehát a (C(1), F(4)) elempártól a (C(7), F(10)) párig teljes a megfelelkezés. Sőt, ilyen esetben van értelme az F(11) elemről is beszélni: az ekvivalenciacsoport hatására az F tömb „megnyúlt”. A FORTRAN nyelv megengedi az EQUIVALENCE

utasítás hatására megnyúlt tömbök ilyen pótlólagos elemeivel való számolást is. Ugyanakkor *tilos egy tömb két különböző elemét ekvivalenciába hozni!*

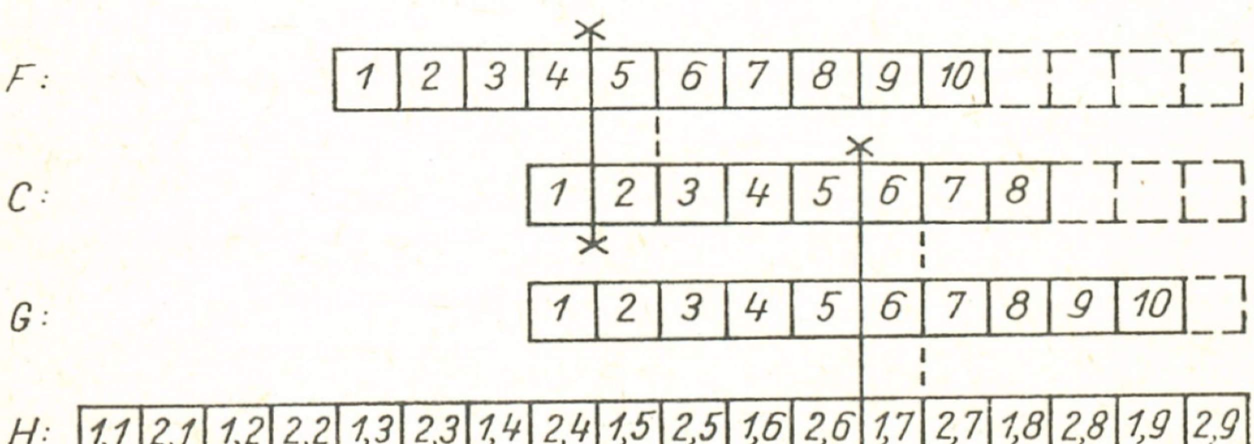
A (4.20) utasítás második ekvivalenciacsoportjának hatását a 4. ábrán mutatjuk be:



4. ábra

Most a H tömb hatására „nyúlik meg" a C és a G tömb.

Végül vegyük észre, hogy a most részletesen taglalt két ekvivalenciacsoport mindegyikében szerepelt a C tömb. De ha a C tömbnek megvan a helye a memóriában, akkor hozzá viszonyítva az F, G és H tömbök is megszabott helyre kerülnek. Tehát pl. F és H elemei közt is létrejön bizonyos megfelelés. Ezért beszélhetünk a két ekvivalenciacsoport együttes hatásáról, melyet az 5. ábrán szemléltetünk.



5. ábra

Ha a (4.19) és (4.20) utasítások összes ekvivalenciacsoportjának hatását összességében vizsgáljuk, a következőket vehetjük észre.

1. A jelzett utasítások formailag öt ekvivalenciacsoportot adnak meg.

2. A formailag különböző ekvivalenciacsoportok között lehetnek olyanok, melyeknek hatása összekapcsolódik. Esetünkben a (4.20) utasítás ekvivalenciacsoportjait a C tömb közvetlenül összekapcsolja. Az ilyen ekvivalenciacsoportokat nem tekintjük logikailag különbözőknek.

Példa bonyolultabb, logikailag nem különböző ekvivalenciacsoportokra, ha a megfelelő tömbdeklarátorok R(2,3), S(2,7), T(14), U(13) és V(4,2):

```
      EQUIVALENCE(R(2,3), T(4)), (U(13), V(1,2)),
     1(T(12), S(2,3), U(2))
```
(4.21)

Ezek* együttes hatását mutatja a 6. ábra.

Az ábrázolás megkönnyítése céljából, most elhagytuk a tömbök „megnyúlásának" szemléltetését.

A továbbiakban az így összekapcsolódó csoportokat együtt *báziscsoportnak* nevezzük. Minden báziscsoportban van legalább egy olyan tömb, amely a báziscsoport hatására a „leghosszabb" lesz: a 6. ábrán ez a memóriában először kezdődő, ezért az ábrán balra kinyúló R tömb. Az ilyen tömb azonosítóját *bázisazonosítónak* nevezzük, mivel a báziscsoport többi elemének a memóriában való elhelyezkedését e tömb kezdetéhez viszonyítjuk. A többi elem ettől mindig jobbra esik.

Eddigi megállapításaink alapján kimondhatjuk, hogy:

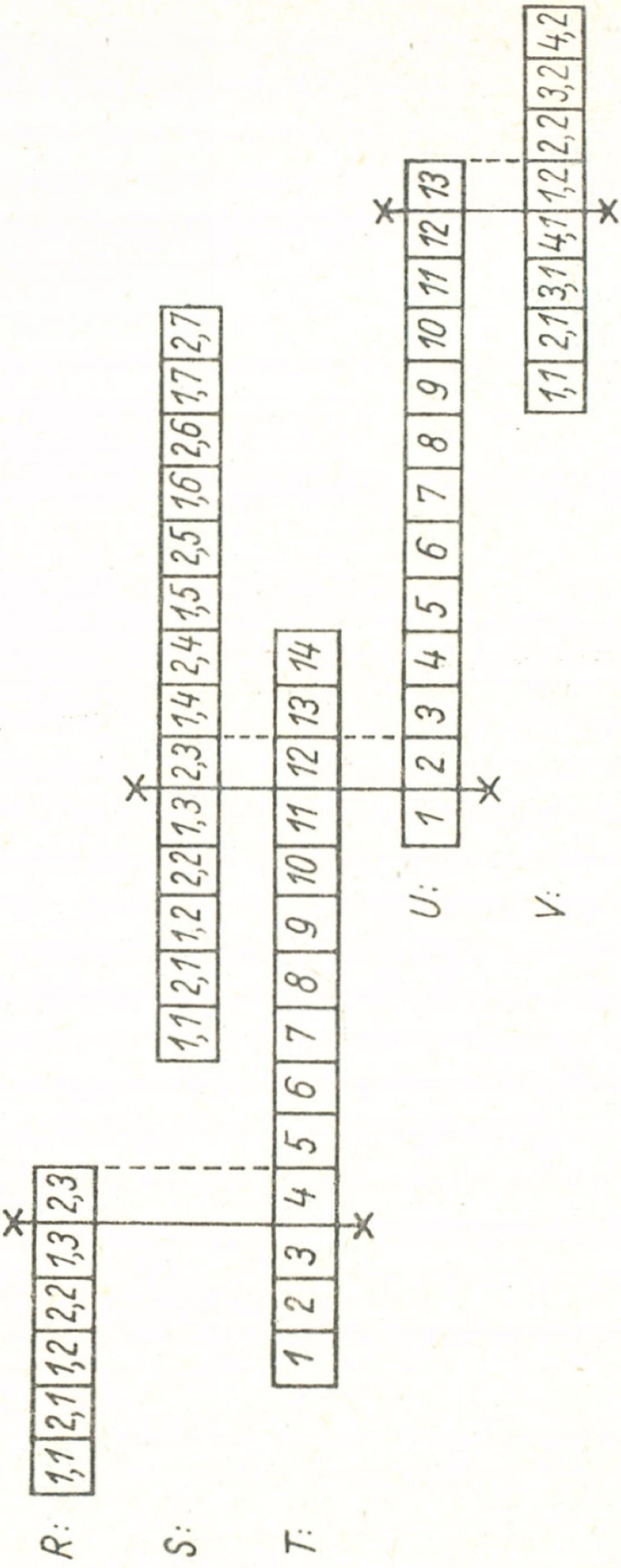
1. *Egy báziscsoport megadható egy vagy több formailag különböző ekvivalenciacsoporttal.*

2. *A különböző báziscsoportokat megadó ekvivalenciacsoportokat logikailag különbözőknek nevezzük.*

3. *Egy szegmens két változója vagy tömbje csak akkor kerülhet ekvivalenciába, ha ugyanahhoz a báziscsoporthoz tartoznak.* (Ez a megállapítás azonban módosul COMMON blokkok esetén, lásd az Ötödik fejezet 5. pontját!)

4. *Egy tömb két különböző eleme nem hozható ekvivalenciába.*

* Tekintettel arra, hogy az utasítás elég hosszú volt, a végét ún. folytatósorban helyeztük el. Ezt jelzi a 6. karakterpozícióban levő 1-es számjegy (lásd az Első fejezet 3. pontját!).



6. ábra

Sőt, most már megadhatjuk az ekvivalenciacsoportok felírására vonatkozó egyetlen szintaktikus szabályt:

5. *Csak olyan ekvivalenciacsoportot írhatunk* be az ekvivalenciautasításokba, *amely nincsen ellentétben sem az előzőleg*

megadott ekvivalenciacsoportokkal, sem pedig önmagával (és — mint az Ötödik fejezet 5. pontjában látni fogjuk — a COMMON utasításokkal sem).

Példák ellentmondásos ekvivalenciacsoportokra: Önmaguknak ellentmondó ekvivalenciacsoportok:

(A(10), B(5), A(3)), (X(5), X(7)) (4.22)

hiszen egy tömb két eleme — mint itt A(10) és A(3) — nem kerülhet ugyanabba a memóriarekeszbe. Egymásnak ellentmondó ekvivalenciacsoportok:

(A(10), B(5)) és (A(13), B(10)) (4.23)

(X(5), Y(8,6)) és (X(3), Y(9,6)) (4.24)

hiszen pl. (4.23) első ekvivalenciacsoportjából az következik, hogy A(13) a B(8) elemmel kerül azonos rekeszbe, nem pedig B(10)-zel, amit a második ekvivalenciacsoport ír elő.

A (4.21) utasítással megadott báziscsoportnak mondanak ellent a következő ekvivalenciacsoportok:

(V(1,1), S(2,3)), (R(1,3), V(2,1)) (4.25)

ami a 6. ábrából azonnal leolvasható. (Ezek az ekvivalenciacsoportok az előzőleg felírt ekvivalenciák összességének mondanak ellent.)

A (4.21) utasítás mellett felesleges felírni a következő, *felesleges* (vagy redundáns) *információt tartalmazó* ekvivalenciacsoportokat:

(R(1,2), T(1)), (V(1,1), S(1,7)) (4.26)

Ezek ugyanis — mint a 6. ábrából látható — (4.21)-ből már következnek.

Megjegyezzük, hogy egyes gépi reprezentációk fordítóprogramjai hibásnak tekintik a FORTRAN programot akkor, ha benne felesleges információt tartalmazó ekvivalenciacsoportot találnak, bár ez az ASA FORTRAN nyelv szabályai szerint nem hiba.

Láthatjuk, hogy az EQUIVALENCE utasítás segítségével annyira ellenőrizhetjük programunk memóriakezelését, ameny-

nyire csak óhajtjuk. De, hogy munkánk még egyszerűbb legyen, a következő könnyítést is megengedik a FORTRAN nyelv szabályai:

6. *Az* EQUIVALENCE *utasításban a többdimenziós tömbök elemeit egy indexszel is írhatjuk. Ilyenkor az index értéke a tömbelem sorszámát jelenti.*

Például a (4.21) utasítással a következő utasítás is egyenértékű:

```
      EQUIVALENCE (R(6), T(4)), (U(13), V(5)),
     1(T(12), S(6), U(2))
```
(4.27)

Jelen pontban meg kell még emlékeznünk az olyan ekvivalenciacsoportokról is, melyek egy memóriarekeszhez különféle típusú mennyiségeket rendelnek. A FORTRAN nyelv szabályai kimondják, hogy ilyen esetben is csak egyféle típusú érték lehet a memóriarekeszben, és csak az ennek megfelelő típusú változóknak, tömbelemeknek van értéke. Az ilyenfajta ekvivalencia kizárólag arra használható, hogy a szegmens egyik részében ugyanezt a memóriarekeszt más részeredmények tárolására tudjuk felhasználni, mint a szegmens másik részében. (Lásd a következő 2. példát.)

Példák más típusú gyakorlati alkalmazásokra:

1. Legyenek az

```
      EQUIVALENCE (BER(1,1), ALAP(1)), (BER(1,2), CSPK(1)),
     1(BER(1,3), VESZP(1)), (BER(1,4), OSB(1))
```
(4.28)

ekvivalenciautasításban szereplő tömbök tömbdeklarátorai:

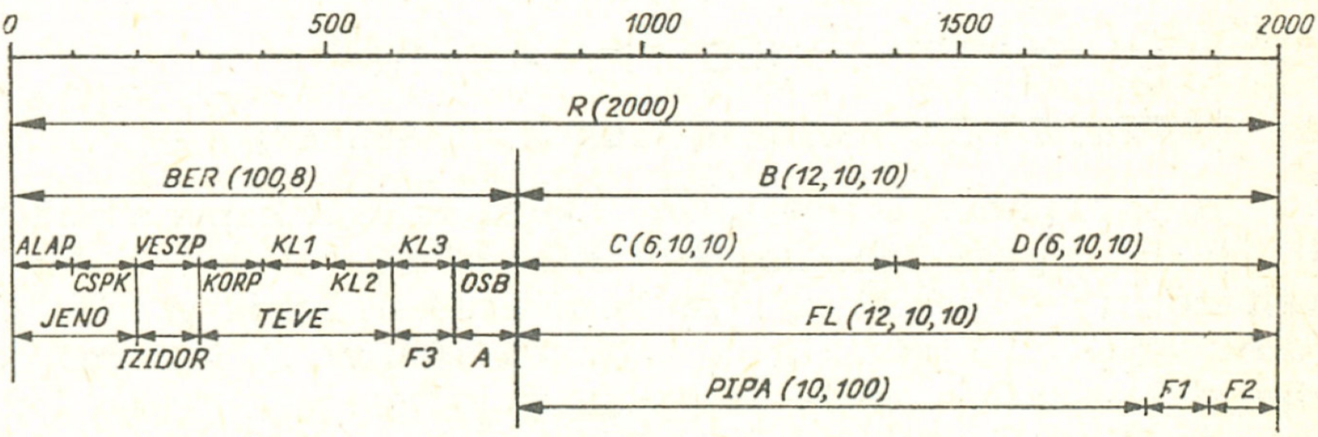
BER(100,4), ALAP(100), CSPK(100), VESZP(100), OSB(100)

Akkor a (4.28) utasítás a 100×4-es bérmátrix egyes oszlopainak ad külön nevet: ALAP (alapbér), CSPK (családi pótlék), VESZP (veszélyességi pótlék), OSB (összbér). Így a programban ezeken a neveken is lehet a mátrixelemekre hivatkozni!

2. Tekintsük a következő deklarációs részt:

```
      INTEGER F1, F2, F3                                        (4.29)
      REAL KL1, KL2, KL3, KORP
      DIMENSION R(2000), A(10,10), B(12,10,10),
     1 FL(12,10,10), F1(10,10), F2(10,10), F3(10,10), KL1(100),
     2 KL2(100), KL3(100), ALAP(100), CSPK(100), VESZP(100),
     3 KORP(100), OSB(100), BER(100,8), C(6, 10,10), D(6, 10,10)
     4 ,IZIDOR(100), TEVE(100,3), JENO(200), PIPA(10,100)
      EQUIVALENCE (R(1), BER(1), ALAP(1), JENO(1)),
     1 (R(101), CSPK(1)), (R(201), VESZP(1)), (R(301), KORP(1)),
     2 (R(401), KL1(1)), (R(501), KL2(1)), (R(601), KL3(1)),
     3 (R(701), OSB(1)), (R(801), PIPA(1), B(1), C(1)),
     4 (R(1401), D(1)), (R(1801), F1(1)), (R(1901), F2(1)),
     5 (R(201), IZIDOR(1)), (R(301), TEVE(1)), (R(601), F3(1)),
     6 (R(701), A(1)), (R(801), FL(1))
```

Az itt deklarált tömböket a nagyszámú ekvivalenciacsoport az R referenciatömb segítségével meghatározott kölcsönös helyzetbe hozza a memóriában, mint azt a 7. ábra is mutatja.



7. ábra

Így igen nagy memóriamegtakarítás érhető el.

Az ábra szerinti elhelyezkedésben csak a JENO, IZIDOR, F1, F2 és F3 tömböknek a többitől eltérő — azaz INTEGER — típusa okozhat esetleg problémát. Különben a memória megfelelő helyén levő számérték azzal az azonosítóval érhető el, amit a számítás menete éppen megkövetel.

5. Kezdőérték adás. A DATA utasítás

Programjainkban előfordulhatnak olyan számadatok, matematikai állandók, melyekkel sokat dolgozik a program és várható, hogy bármikor is kívánjuk a programot felhasználni, értékük nem fog megváltozni. Ilyen számadat lehet például a $\pi = 3{,}14\ldots$, vagy az $e = 2{,}71\ldots$ szám stb.

A FORTRAN nyelvnek külön utasítása van arra, hogy bizonyos változóknak ilyen célból értéket adjon. Ennek a segítségével például a PI változónak adhatom a 3.141592653589793238 számértéket, mint *kezdőértéket*, és a továbbiakban a képletekben előforduló PI változó ezt a számértéket fogja jelenteni. Az utasítás alakja:

$$\text{DATA } l_1, l_2, \ldots, l_N \tag{4.30}$$

ahol az $l_1, l_2, \ldots, l_N$ listaelemek speciális szerkezetű kettős listák; egy ilyen l_k listaelem alakja a következő:

$$r_1, r_2, \ldots, r_{M_k}/d_1, d_2, \ldots, d_{M_k}/ \tag{4.31}$$

Az r_l és d_l listaelemek száma egyenlő* és kölcsönösen megfelelnek egymásnak. Az r_l listaelemek indexes vagy index nélküli változók, a két ferde vonal közé zárt d_l listaelemek pedig a nekik rendre megfelelő kezdőértékek. Például:

```
      DATA KETTO, NEGY, PI/2,4,3.14159/,
     1FEL, OTOD, HATOD/0.5,0.2,0.16666667/
```
(4.32)

A (4.32) utasítás hatására a 2 kezdőértéket kapja a KETTO INTEGER típusú változó, a 4 kezdőértéket kapja a NEGY változó, s í. t., végül az 1/6 kezdőértéket kapja a HATOD REAL típusú változó (ha a változók típusát az automatikus típusdeklarációs szabály szabja meg).

A „kezdőérték" kifejezés itt azt jelenti, hogy a program lefordítása után mindaddig, míg a program értéküket meg nem változtatja, a változók értéke a megfelelő kezdőérték lesz; illetve a változónak megfelelő memóriarekeszben a kívánt érték található. (Olyan számológépek esetén, amelyekbe a célprogramokat futtatás előtt „betöltik" vagy „betáplálják" a memóriába,

* Az ismétlési tényezők használatától a könnyebb megértés kedvéért egyelőre eltekintünk. Módosító hatásukat alább tárgyaljuk.

jogosabb lenne talán a *betöltési érték* kifejezés használata.) Maga a DATA utasítás azonban nem zárja ki azt, hogy a programban később pl. egy értékadó utasítással a kezdőértékkel rendelkező változó értékét megváltoztassuk.

Mint a (4.32) utasítás is mutatja, a ferde vonalak közt *felsorolt konstansok típusa megegyezik a megfelelő változók típusával*. Ezenkívül megadhatunk szövegkonstanst is (lásd a következő *a)* alpontot).

Szabad ugyanazt a kezdőértéket több változónak is adni, például:

```
DATA A(1), A(2), B, BB/4.5, 2.5, 2.5, 2.5/        (4.33)
```

A FORTRAN nyelv megengedi, hogy ezt az utasítást a következő egyszerűsített jelöléssel írhassuk fel:

```
DATA A(1), A(2), B, BB/4.5, 3*2.5/                (4.34)
```

azaz szabad a konstansok listájában ismétlési tényezőket alkalmazni.

Ügyelni kell arra, hogy a DATA utasítások ne mondjanak ellent EQUIVALENCE utasításoknak! Egy memóriarekeszben csak egyetlen kezdőérték tárolható, de viszont az ekvivalencia-utasítás két vagy több változóhoz ugyanazt a memóriarekeszt rendelheti. E változók közül csak egynek adhatunk kezdőértéket, ami egyben az összes ugyanilyen típusú ekvivalenciába hozott változónak is közös kezdőértéke lesz. Elemezzünk néhány deklarációs részt ebből a szempontból!

1. Tekintsük a következő deklarációs részt:

```
      LOGICAL L(5), L1(10)
      INTEGER IA(15), IB(15)
      REAL A(15), B(15)
      EQUIVALENCE (L(1), IA(1), A(1)),                (4.35)
     1(L1(1),IA(6)), (IB(1), B(1))
      DATA L(1), L(2), IA(3), IA(4), A(5), A(6)/.TRUE.,
     1.FALSE., 1,2,3.0,4.0/
```

Itt 2—2 logikai, egész és valós tömb deklarációját látjuk. A felírt három ekvivalenciacsoport két báziscsoportot definiál: az első báziscsoportba tartozik az A, IA, L és L1 tömb. Közülük csak az L tömb „nyúlik meg". A második báziscsoportba az egymással ekvivalens, de különféle típusú B és IB tömbök tartoznak.

Az első báziscsoport első hat memóriarekeszébe egy DATA utasítás kezdőértékeket tesz. Az utasítás formailag helyes, hiszen a változók és a konstansok típusa egymásnak megfelel. Látunk itt példát logikai kezdőérték adásra is. De a DATA utasítás nincs ellentmondásban az EQUIVALENCE utasítással sem, hiszen egy rekeszbe csak egy kezdőérték kerül. Például az L(1), IA(1) és A(1) indexes változók közül a LOGICAL típusú L(1) tömbelemnek van TRUE kezdőértéke. Amíg a program futása során IA(1) vagy A(1) értéket nem kap, a logikai tömbelemnek lesz értéke és az aritmetikaiaknak nem lesz. Például, ha végrehajtásra kerül az

```
| |IA(1)=10
```

utasítás, akkor L(1) *értéke* ezzel egyidőben *elvész* és egy egész értéket fog a memóriarekesz őrizni. Ha más egész típusú változó is ekvivalens lenne IA(1)-gyel, az is megkapná most a 10 értéket. Az A(1) valós típusú változónak természetesen még most sincs értéke.

2. Másik példa:

```
| |LOGICAL FI(10)
| |INTEGER IA(15), IB(15)
| |REAL A(10), AA(15), B(20), BB(15)
| |EQUIVALENCE (FI(2), IA(1), A(1), AA(1)),        (4.36)
|1|(AA(15), B(5)), (B(6),BB (1), IB(1))
| |DATA FI(7), FI(8), FI(9), FI(10), IA(5), IA(6),
|1|A(11), A(12)/4*.TRUE., 2*1, 2*1.0/
```

Látható, hogy ismétlési tényezőket bármilyen típusú konstans előtt használhatunk. Most egyetlen báziscsoport van meg-

adva három ekvivalenciacsoport segítségével, melyek nem tartalmaznak felesleges információt; a bázisazonosító a FI logikai tömbazonosító. A DATA utasításban *hiba van:* FI(7)-nek és IA(6)-nak nem lehet egyszerre kezdőértéket adni, hiszen hozzájuk azonos memóriarekesz tartozik!

Egyes gépi reprezentációk megengedik, hogy tömbazonosítók is előforduljanak a DATA utasítás változólistáján. Ilyenkor a tömbazonosító megfelel az összes sorrendbe szedett eleméből felírható listának. Ez a lehetőség az ASA FORTRAN-ban nincs meg.

Az EQUIVALENCE és DATA utasításokkal kapcsolatos további szabályokkal a COMMON utasítással kapcsolatban fogunk megismerkedni (lásd Ötödik fejezet 5. és 6. pontot).

a) Szövegkonstansok

Eddig találkoztunk különféle típusú számkonstansokkal, s a két logikai konstanssal (.TRUE. és .FALSE.). De a számítások során felmerülhet annak a szükségessége is, hogy a program betűkből vagy számokból — azaz karakterekből — szövegeket állítson össze, ilyen szövegeket a számítási eredmények között kiírjon, vagy az adatok egyeztetése és kezelése kedvéért velük együtt szövegeket beolvasson. Ezeket a szövegeket szokás *stringeknek* is nevezni, bár a FORTRAN nyelvben a *Hollerith konstans* elnevezés a szokásos.

A szövegek tárolásához a FORTRAN nyelvben nem kell különleges, pl. szöveg-típusú változó, hanem bármiféle eddig tárgyalt típusú változó felhasználható erre a célra. Hogy a memóriakihasználás gazdaságosabb legyen, az egyes gépi reprezentációkban egy változó több karaktert tárolhat. Ezt lehetővé teszi, hogy a gépi jelkészlet ábrázolására a számológépek 6—8 db kettes számrendszerbeli számjegyet — ún. *bitet* — használnak, míg egy változó ábrázolására szolgáló memóriarekesz 24, 36 vagy 48 bitből állhat. Ezek szerint 4, 6 vagy 8 karaktert tartalmazhat a változó. Az egy változóban tárolható karakterek számát a továbbiakban *g*-vel jelöljük és *elemszélességnek* nevezzük (lásd a Hatodik fejezet 4*g* alpontját). A karaktereket balról jobbra számozzuk. Ilyenkor azt mondjuk, hogy a *változónak karakter típusú értéke* van.

Az egyes gépi reprezentációkban az ASA FORTRAN-ban nem szabályozott módon értelmezhető a változó „tartalma", mint karaktersorozat. Az értelmezés szabályait, illetve a karaktereket a megfelelő számológép karakter- és számábrázolási szabályai szabják meg.

Felmerül a kérdés, hogyan vehet fel egy változó karakter típusú értéket. Ennek egyik módja az, hogy DATA utasításban a változónak karakter típusú kezdőértéket adunk. Erre is használhatók a FORTRAN nyelv szövegkonstansai. Alakjuk:

$$n\mathrm{H}h_1\, h_2 \ldots h_n \qquad (4.37)$$

ahol

n pozitív egész szám, mely megadja a szövegkonstans hosszát, vagyis karaktereinek számát;

H jelzi, hogy utána kezdődik a szöveg, vagyis a Hollerith konstans;

$h_1\ h_2\ \ldots\ h_n$ pedig karakterek n elemű sorozata. A betűköz is karakternek számít. Itt általában nemcsak a FORTRAN jelkészlet karaktereit szabad használni, hanem a megfelelő számológépben ábrázolható összes karaktert. Például:

```
8HFORTFILE                                   (4.38)
12HPISZKOS▽FRED                              (4.39)
11HHOVANSCSINA                               (4.40)
22HA▽FAJHO▽1-ES▽TABLAZATA                    (4.41)
```

Tegyük fel, hogy g értéke pl. 8. Értelmezzük akkor a következő DATA utasítást:

```
  DATA A(8), A(6), A(4), A(1)/8HFORTFILE,
 112HPISZKOS▽FRED, 11HHOVANSCSINA,              (4.42)
 222HA▽FAJHO▽1-ES▽TABLAZATA
```

Először A(8) felveszi a FORTFILE szöveg értéket. Ezután A(6)-ba bekerül a PISZKOS▽ szövegrész, de a FRED szöveg az A(7) tömbelembe csúszik át és 4 db szóközzel egészítődik ki: A(7) tartalma: FRED▽ ▽ ▽ ▽. Hasonlóképpen alakul ki az A tömb többi elemének tartalma is:

Tömbelem	Tartalom
A(4)	HOVANSCS
A(5)	INA▽▽▽▽▽
A(1)	A▽FAJHO▽
A(2)	1-ES▽TAB
A(3)	LAZATA▽▽

Látjuk tehát, hogy bár csak 4 db indexes változó szerepel, a (4.42) utasításban, 8 db kapott szöveg típusú kezdőértéket, mivel egyes szövegek hossza meghaladta a $g=8$ értéket. Ami az egyes változókból, mint szabad hely kimaradt volna, oda a szükséges számban betűköz karakterek kerültek.

Bár a szabványos ASA FORTRAN nyelv ezt a kérdést nem szabályozza, sok gépi reprezentációban külön eljárásokat dolgoztak ki arra, hogy a változók belsejéből tetszőleges sorszámú karaktert ki tudjanak emelni, a karaktereket átcsoportosítsák stb. Ezek a karaktermanipulációk például az ICT FORTRAN-ban jól ki vannak fejlesztve. Ebben a könyvben a szövegkonstansokkal a továbbiakban a CALL utasítás (lásd az Ötödik fejezet 3. pontját) és az INPUT-OUTPUT utasítások (lásd a 6. fejezetet) kapcsán fogunk még találkozni.

Végül egy gyakorlati tanács: programjaink írásakor ügyeljünk arra, hogy lehetőleg ne nyúljon át szövegkonstans a következő sorba: inkább adjunk meg több rövidebb szövegkonstanst. Egyes gépi reprezentációkban ugyanis a folytatósorokba való áttérés kérdése nincs ilyen esetben tökéletesen megoldva!

DATA utasításban a szövegkonstansok más típusú konstansokkal tetszőlegesen keverhetők. Továbbá, ha egy változó szövegkonstans értéket kapott, az összes vele ekvivalenciába hozott bármilyen típusú változó is ugyanazt a szövegkonstanst veszi fel értékül.

6. Utasításfüggvények

A (3.39) összefüggés szerint X kis értékei esetén a ch(X) függvényt a következő módon is ki lehet számítani:

```
  CH(X) = 1.0+X**2*(0.5+X**2*(1.0/24.0+
1 X**2/720.0))                                  (4.43)
```

Tegyük fel, hogy szegmensünkben több helyen van szükség arra, hogy ezt a függvényt használhassuk. Például szeretnénk leírni a következő utasításokat:

```
A = CH(B)
W = A**2
C(13) = CH(F-5.0)+CH(F+5.0)        (4.44)
C(14) = A-C(13)
⋮
```

Most azt akarjuk, hogy (4.44) utasításaiban CH a (4.43)-nak megfelelő függvényt jelentse ugyanúgy, ahogy a COS azonosító standard külső függvényt azonosít. A FORTRAN-ban erre valók az ún. *utasításfüggvények*. Például, ha a deklarációs rész végére felírjuk a (4.43) utasítást, a szegmens belsejében (4.44)-gyel a kívánt jelentést érhetjük el. (4.43) egyenlőségjelének bal oldalán szerepel a CH *utasításfüggvény azonosító*, melyet zárójelbe zárva követ a független változó. Ez itt „csak" *formális paraméter*, hiszen a (4.44) programrészletben változó azonosító vagy kifejezés helyettesíti, melyeket *aktuális paramétereknek* nevezünk. A formális és aktuális paraméterek fogalmával az Ötödik fejezetben részletesen foglalkozunk.

A FORTRAN nyelvben az utasításfüggvényeket a deklarációs rész végén lehet

$$f(a_1, a_2, \ldots, a_n) = e \tag{4.45}$$

alakú utasításokkal megadni, ahol f az utasításfüggvény azonosítója; az a_I-k az ún. formális paraméterek, melyeknek száma legalább egy. A formális paraméterek tetszőleges nemindexes változó azonosítók lehetnek. Szerepük kettős: 1. megadják a paraméterek típusát és sorrendjét; 2. szerepelhetnek az e kifejezésben. Az e kifejezés és az f utasításfüggvény típusa olyan kell, hogy legyen, ami megfelel az

$$f = e$$

értékadó utasítás végrehajthatósági feltételeinek (lásd a 6. táblázatot). Az e kifejezésben szerepelhet nem szövegtípusú konstans, belső és külső standard függvény azonosítója, sőt előzőleg definiált utasításfüggvény azonosítója is más függvényeljá-

rások (lásd az Ötödik fejezet 3. pontját) azonosítóival együtt, továbbá a szegmens minden olyan index nélküli változója, melynek azonosítóját nem használtuk formális paraméter azonosítására.

Az utasításfüggvényeket ugyanúgy használhatjuk a szegmensben, mint a belső standard függvényeket. Az *aktuális paraméterek típusának* pedig *meg kell egyezniük a megfelelő formális paraméterek típusával.*

Példák:

1. A (4.43) utasításfüggvényt adja meg például a következő definíció is.

```
      F(X) = X**2
      CH(X) = 1.0+F(X)*(0.5+F(X)*(1.0/24.0+            (4.46)
     1F(X)/720.0))
```

2. A háromszög területét az oldalak függvényében például a következő utasításfüggvényekkel is definiálni lehet:

```
      S(X1,X2,X3) = (X1+X2+X3)/2.0
      TER(X1,X2,X3) = SQRT(S(X1,X2,X3)*(S(X1,X2,X3)-
     1X1)*(S(X1,X2,X3)-X2)*(S(X1,X2,X3)-X3))
```

3. Bevezethetünk saját célra logikai utasításfüggvényeket is, például:

```
      LOGICAL EQI, L1,L2
      EQI(L1,L2) = (L1.AND.L2).OR.(.NOT.L1.AND..
     1NOT.L2)
```

4. Egy példa arra, hogy különböző típusú argumentumokat is lehet használni:

```
      DOUBLE PRECISION DBL,Y
      DBL(Y,N) = (Y**N+Y**(-N))/A(2)
```

7. Gyakorlatok

1. A Harmadik fejezet gyakorlatainak megoldásait lássuk el megfelelő deklarációs résszel. Csak képletesen jelölve az INPUT-OUTPUT utasításokat („beolvasás", „kiírás"), az egyes programrészletekből állítsunk össze teljes programokat!

2. Írjunk ekvivalenciautasításokat, melyeknek hatására:
a) a 100 elemű A és B tömbök egymás mögé kerülnek a memóriában,
b) úgy kerülnek egymás mögé, hogy közöttük további 100 tömbelem számára marad hely, tehát A esetleges kisebb „megnyúlása" nem „rontja el" B elemeinek értékeit!
Egészítsük ki ekvivalenciautasításainkat a szükséges dimenzionáló utasításokkal!

3. Keressünk hibákat a következő deklarációs részekben:

a)
```
  |DOUBLE PRECISION DA,DB
  |DATA A,B,DA,DB/3.1415926,2*
 1|2.71,3.14159265358979323846/
```

b)
```
  | DIMENSION A(2,5,10), IA(10,10)
  | EQUIVALENCE (A(50), IA(50)), (A(1), B(1))
  | DATA A(12), IA(2,2)/125,8HCINKEFULKE/
```

c)
```
  | LOGICAL FE(10), FJ(20)
  | DIMENSION IA(50)
  | FE(X) = X**2+IA(12)
  | EQUIVALENCE (IA(11), FE(1)),
 1| (IA(21), FJ(1))
  | DATA FJ(2)/8H▽.TRUE.▽/
```

Írjunk, ha lehet, helyes deklarációs részeket a hibásak helyett.

4. Írjunk utasításfüggvényt, mely
a) a paralelogramma területét és kerületét számítja ki egy szögből és két oldalból;
b) a szabályos testek (tetraéder, hexaéder, oktaéder, dodekaéder, ikozaéder) felszínét és térfogatát számítja ki egy oldalélből;
c) a 4 valós formális paraméter közül a legkisebbet vagy a legnagyobbat veszi fel aktuális értékül;
d) egyetlen argumentumát a félszög tangensének tekinti és az egész szög trigonometrikus függvényeit adja eredményül.

Ötödik fejezet

TÖBB SZEGMENSBŐL ÁLLÓ FORTRAN PROGRAMOK

1. Bevezetés

Könyvünk előző fejezeteiből merített ismereteink alapján most már képesek vagyunk — a ki- és bevitel műveleteitől eltekintve — mindazt FORTRAN-ban leírni, ami e nyelvben egyáltalán leírható. Úgy tűnik tehát, hogy nincs más hátra, mint a ki- és bevitel elsajátítása, és aztán bátran nekivághatunk akármilyen bonyolult programok megírásának.

Elvileg ez így is van, ha azonban megkísérelnénk egy valamennyire is összetett feladatot úgy beprogramozni, hogy nem bontanánk szét kisebb részfeladatokra, akkor hamarosan szemben találnánk magunkat több előre nem látott nehézséggel. Először is, egy ilyen egyetlen részből álló mammut-program nehezen áttekinthető, működését nem lehet nyomon követni, ezért funkcionális ellenőrzése („belövése") igen hosszadalmas. Tekintve, hogy az idő pénz (főként, ha gépidőről van szó), ez önmagában is nyomós érv amellett, hogy a nagy programokat mindig célszerű valahogyan feldarabolni.

Egy másik szempont, hogy a legtöbb gyakorlati programozási feladat visszavezethető a numerikus analízis valamelyik alapfeladatára (differenciálegyenlet megoldása, mátrixalgebra és aritmetika, transzcendens egyenlet megoldása stb.), vagy ilyen alapfeladatok kombinációjára. Mivel szinte minden számolóközpontban találhatók ezeknek az alapfeladatoknak a megoldására irodalmilag feldolgozott és számtalan változatban beprogramozott eljárások, semmiképpen sem volna gazdaságos ezek megírását minden esetben újra kezdeni. A megoldás kézenfekvő: a kész eljárásokat át kell venni és „előregyártott elem"-ként beépíteni a készülő programba. Ezzel mind a program megírásakor, mind a kipróbáláskor időt takaríthatunk meg, feltéve, hogy a beépített eljárás már előzőleg alaposan ellenőrizve volt.

Ezzel kapcsolatban azonban jelentkezik egy nehézség is. Tegyük fel, hogy előttünk valaki megírt egy FORTRAN programot, amely négyzetes mátrixok invertálására alkalmas, mégpedig 25×25-ös mátrixokra készítette és a mátrixot A-val, az inverzet AINV-vel jelölte. Nekünk ez a program tulajdonképpen megfelelne, de feladatunkban a mátrix rendszáma nem 25, hanem 20, neve nem A, hanem X, az inverz azonosítója pedig nem AINV, hanem Y. Hogyan tudjuk ezt a programot a mi feladatunkra adaptálni?

Elvileg igen egyszerűen: nem kell egyebet tennünk, mint átjelölnünk az egészet A-ról X-re, AINV-ről Y-ra és 25-ről 20-ra. Tegyük fel, hogy három hibát vétettünk az átíráskor; ebből kettő kiderült az összeolvasás során, a harmadik „átcsúszott". Majd elkészül az egész program és egy próbafeladatot hibásan végez el. Mit lehet tenni: az invertáló program működését nem ismerjük pontosan, ezért felkeressük a szerzőt, aki a program írása óta eltelt két év alatt régesrégen elfelejtette az egész eljárás logikáját, de azért néhánynapos fejtörés (és esetleg néhány gépi próba) után megmutatja azt az egyetlen változót, amelyet egyetlen helyen mégiscsak elmulasztottunk átjelölni.

Ez az út nyílván nem járható. Különböző helyeken és időben készült programrészeknek egyetlen programmá való egyesítése csak akkor lehet megfelelő hatásfokú, ha az egyes részleteknek saját hatáskörükön belül önálló jelölésrendszerük van, és ezt nem kell az összeépítés közben kézileg átalakítani. Példánkból láthatóan az ilyen előregyártott programok hajlékonyságához az is szükséges, hogy bizonyos paramétereket, mint pl. a mátrix rendszáma, változóként lehessen megadni.

A fentebb felsorolt programozási és programozói igényeket kívánja kielégíteni a FORTRAN nyelvben a programok *szegmentálásának* lehetősége. Ennek a lehetőségnek főbb jellemzői a következők:

1. A program minden egyes szegmense önálló egészet alkot: a program többi szegmensének jelölésrendszerétől független, saját jelölésrendszere van. Ugyanaz az azonosító különböző szegmensekben szabadon felhasználható különböző programozási objektumok jelölésére.

2. A FORTRAN rendszer biztosítja a programozó által

előírt programozási objektumok (pl. változók) rendszeres „belső" átjelölését. Ez azt jelenti, hogy a programozó előírhatja, egy adott eljárást milyen nevű változókon vagy tömbökön kell végrehajtani (átjelölés A-ról X-re és AINV-ről Y-ra stb.).

3. Fordítás szempontjából az egyes szegmensek teljesen függetlennek tekinthetők. Ez az ún. *szegmensenkénti fordíthatóság elve*, amely azt mondja ki, hogy amikor a fordítóprogram valamely FORTRAN szegmenst fordít, nem tartja nyilván, hogy előzőleg mely más szegmens fordításával foglalkozott és ezután milyen szegmensek következnek; az adott szegmens fordításához más szegmensekből semmiféle kiegészítő információt nem vehet elő. Az egyes szegmensek összeállítását egyetlen végrehajtó programmá a fordítóprogram külön menetben végzi, sőt, egyes esetekben ezt a fordítóprogramtól különböző másik program hajtja végre.*

4. A program egyes szegmensei közti kapcsolatot az ún. *szegmensközi azonosítók* biztosítják. Ezek olyan speciális azonosítók, amelyekre más szegmensekben is lehet hivatkozni. Ilyenek a szegmensek nevei, valamint az ún. *közös adatmezők* (COMMON blokkok) nevei.

E fejezet további pontjaiban a programszegmentálás kérdéseit ismertetjük. A következő 2. pont a szegmentált programok formai felépítésével foglalkozik. Ezt követőleg a 3. pontban a programszegmensek említett „átjelölési" funkcióit (paraméterátvétel, paramétercsere) tárgyaljuk, majd a deklaráló utasítások használatát érintő olyan szabályokat ismertetünk, amelyeknek csak több szegmensből álló programok esetén van jelentőségük (4—6. pontok). A 7. pontban egy újabb deklaráló utasítást (EXTERNAL) ismerünk meg, amelyet ugyancsak szegmentált programokban használunk.

* A FORTRAN fordító a lefordított szegmenseket ún. *áthelyezhető bináris* alakban állítja elő, a fordítás végső lépése ezeknek a szegmenseknek teljes programmá való összeállítása. Az áthelyezhető bináris formátum géptől függő, és általában ez a célnyelve a gép valamennyi fordítóprogramjának. Az összeállító program segítségével különböző forrásnyelvekből (pl. ALGOL-ból és FORTRAN-ból) fordított programszegmensek is összeszerkeszthetők.

2. Szegmensnyitó és -záró utasítások. Szegmentált programok szerkezete

Egy teljes FORTRAN program egy vagy több programszegmensből állhat. Minden egyes szegmens szegmensnyitó utasítással kezdődik, és szegmenszáró utasítással végződik. A szegmensnyitó utasítások az ASA FORTRAN-ban a

SUBROUTINE (5.1)

és a

FUNCTION (5.2)

utasítások. (Van egy harmadik, speciális szegmensnyitó utasítás is, a BLOCK DATA, ezzel a 6. pontban foglalkozunk.) A szegmenszáró utasítás az

END (5.3)

utasítás. Az END utasítás nem folytatható folytatósorban, pl. nem megengedett a következő írásmód*:

```
|  |EN
| X|D
```

A SUBROUTINE és a FUNCTION utasítások szintaxisát egyelőre nem részletezve, egy több szegmensből álló program szerkezete a következő:

```
SUBROUTINE S1
      ⋮
END
SUBROUTINE S2                (5.4)
      ⋮
END
SUBROUTINE S3
      ⋮
END
```

* Itt a 6. karakterpozícióban levő X betű a folytatósort jelöli. Hadd emlékeztessünk itt a 13. oldalon kimondott 3. formai szabályra, mely e célra megengedi betűk tetszőleges felhasználását is!

Itt S1, S2, S3 az egyes szegmensek azonosítói. Mindegyik szegmens a SUBROUTINE utasítástól az END utasításig terjed. Példánk egy három szegmensből álló programot mutat. (5.4)-ben bármelyik SUBROUTINE utasítás helyett állhat FUNCTION (vagy BLOCK DATA, lásd később) utasítás is.

Mint már említettük és (5.4)-ben láttuk, minden egyes szegmens egy szegmensnyitó utasítással kezdődik. Ezt a szegmens deklarációs része követi az utasításfüggvények deklarációival együtt (ha ilyen van), és végül a végrehajtható utasítások. A szegmenst az END utasítás zárja le. Az itt leírt szerkezet mindenfajta szegmensre érvényes, legfeljebb bizonyos összetevők hiányozhatnak.

Gépi reprezentációs nyelvekben definiálni szoktak egy negyedik szegmensnyitó utasítástípust is, amely a program ún. fő szegmensének élén áll. Ennek az utasításnak a neve lehet gépi reprezentációtól függően PROGRAM, MASTER, MAIN stb. A fő szegmens jelentőségére később visszatérünk.

Áttérünk most a szegmensnyitó utasítások szintaxisának tárgyalására. Először a SUBROUTINE utasítással foglalkozunk. Az utasítás alakja:

SUBROUTINE S(A1, A2, ..., AN) (5.5)

vagy

SUBROUTINE S (5.6)

A kerek zárójelek közé zárt A1, A2, ..., AN lista az ún. *formális paraméterek* listája. A listaelemek tetszőleges azonosítók lehetnek. Mint az (5.6) változatból látható, a lista lehet üres, amely esetben a kerek zárójelek is hiányoznak. „S" a szubrutin neve (azonosítója); szintén tetszőleges azonosító.

A FUNCTION utasítás alakja:

FUNCTION F(A1, A2, ..., AN) (5.7)

vagy

t FUNCTION F(A1, A2, ..., AN) (5.8)

ahol F a szegmens neve (az ilyen szegmenst nem szubrutinnak, hanem függvénynek nevezzük); A1, A2, ..., AN pedig formális paraméterek, akárcsak az (5.5) utasításban. F és az AI paraméterek tetszőleges azonosítók lehetnek. (5.8)-ban *t* a típusdeklaráló alapszavak — REAL, INTEGER, COMPLEX, DOUBLE

PRECISION vagy LOGICAL — valamelyike. Mint (5.7) mutatja, ez a típusdeklaráló alapszó hiányozhat a szegmensnyitó utasításból. A SUBROUTINE utasítással ellentétben azonban a FUNCTION utasításban nem hiányozhat a formális paraméterek listája! *A* FUNCTION *utasításnak mindig kell hogy legyen legalább egy formális paramétere.* Ennek a szabálynak az okát a későbbiekben értjük majd meg.

Példák a szegmensnyitó utasításokra:

SUBROUTINE GYUFA(X, Y, W, Z) (5.9)

SUBROUTINE NAMIEZ (5.10)

FUNCTION VAZMI(EZ, AZ) (5.11)

INTEGER FUNCTION CSORT(I) (5.12)

Példáink rendre az (5.5), (5.6), (5.7), (5.8) lehetőségek egy-egy illusztrációját adják. Amint az (5.12) példából látható, a formális paraméterek listája — bármilyen típusban — lehet egyelemű, amely esetben természetesen az elválasztó vessző elmarad.

Az eddigiekben a FORTRAN programok szerkezetét kizárólag sztatikusan vizsgáltuk. Az egyes programszegmensek dinamikus kapcsolatára igen sok szabály vonatkozik, amelyeket a következő ponttól kezdve, fokozatosan tekintünk át.

3. Szubrutinok és függvények aktivizálása. Aktuális és formális paraméterek

a) SUBROUTINE szegmensek. A CALL és a RETURN utasítás

Mielőtt a szubrutin szegmensek használatának szabályaira rátérnénk, bemutatunk azokra néhány gyakorlati példát.

Példák

1. Tegyük fel, hogy olyan programot akarunk írni, amelyben többször és különböző helyeken kell bizonyos változókon síkbeli ortogonális transzformációt végezni. (Szemléletesen: a síkbeli derékszögű koordinátarendszert φ szöggel kívánjuk el-

forgatni és sok pont új koordinátáira van szükségünk.) Mint ismeretes, valamely x és y skaláris változókra alkalmazott síkbeli ortogonális transzformáció a következő képletek alapján hajtható végre:

$$u = x \cos \varphi + y \sin \varphi \qquad (5.13)$$

$$v = -x \sin \varphi + y \cos \varphi.$$

Olyan FORTRAN programszegmensre lenne tehát szükségünk, amely ezt az átalakítást könnyen, lehetőleg kevés adminisztrációs utasítással végzi el, különböző transzformálandó változókon és különböző elforgatási szögekre. Ilyen FORTRAN szegmens lehet a következő:

```
SUBROUTINE ORTOG (X, Y, U, V, FI)
U = X*COS (FI)+Y*SIN (FI)
V = -X*SIN (FI)+Y*COS (FI)          (5.14)
RETURN
END
```

Tegyük fel most, hogy ortogonális transzformációt kívánunk végezni a P és Q programbeli változókon, α szöggel, és a transzformált változókat a programban R-rel és S-sel jelöltük. Hogyan végezhető el ez a transzformáció az (5.14) eljárás segítségével?

A megoldás: programunkban leírjuk a következő „*szubrutinhívó*" utasítást:

```
CALL ORTOG (P, Q, R, S, ALFA)          (5.15)
```

Ennek hatására az (5.14) eljárás (szubrutin) *aktivizálódik*, azaz végrehajtódik az (5.15) CALL utasításban megadott ún. *aktuális paraméterekkel*. A szubrutin végrehajtásakor az aktuális paraméterek a formális paraméterek helyére lépnek, vagyis a szubrutin úgy működik, mintha annak minden egyes formális paramétere helyett a megfelelő aktuális paraméter állna. Esetünkben ez azt jelenti, hogy az (5.14) szubrutin végrehajtásakor X helyébe P, Y helyébe Q, U helyébe R, V helyébe S, és végül FI helyébe ALFA helyettesítődik. A gép tehát az (5.15) CALL

utasítás alapján a következő utasításokkal megegyező hatású műveleteket végez:

$$\begin{aligned} R &= P*COS(ALFA)+Q*SIN(ALFA) \\ S &= -P*SIN(ALFA)+Q*COS(ALFA) \end{aligned} \qquad (5.16)$$

(a RETURN utasítás jelentésére később visszatérünk).

Előfordulhat, hogy ugyanabban a programban, amelyben az imént az ORTOG eljárást aktivizáltuk, egy másik helyen is szükség van ortogonális transzformációra, azonban ezúttal nem P, Q, R, S és ALFA paraméterekkel, hanem (rendre) A-val, B-vel, C-vel, D-vel és Θ-val. Eljárásunk most is változatlanul alkalmazható, *csupán az aktivizáló* CALL *utasítást kell megváltoztatni:*

CALL ORTOG (A, B, C, D, THETA) (5.17)

(5.17) hatására a következő FORTRAN utasításokkal egyenértékű műveletek elvégzésére kerül sor:

$$\begin{aligned} C &= A*COS(THETA)+B*SIN(THETA) \\ D &= -A*SIN(THETA)+B*COS(THETA) \end{aligned} \qquad (5.18)$$

Aktuális paraméterként nemcsak változót, hanem kifejezést is megadhatunk, amennyiben annak típusa megegyezik a megfelelő formális paraméter típusával, és amennyiben ez nem vezet értelmetlenségre. Megengedett például a következő CALL utasítás:

CALL ORTOG(A1 * * 2 – A2 * * 2, A1 + A2, R, S, BETA) (5.19)

(5.14)-ben U, V és FI, jellegüket tekintve, „kimenő" (azaz eredményt megadó) paraméterek, ezért ezeknek értelemszerűen nem feleltethetünk meg kifejezés-paramétert. Ugyanakkor a két „bemenő" paraméter, X és Y helyett — mint a példában is tettük — írhatunk aritmetikai kifejezést, és ez nyilván azt jelenti, — példánknál maradva — hogy az ORTOG eljárás X = A1 * * 2 – A2 * * 2-tel és Y = A1 + A2-vel hajtódik végre.

Az előzők alapján nyilvánvaló a paraméterek „formális", ill. „aktuális" jelzője. A szubrutin szegmens sohasem a formális paraméterekkel, hanem mindig a helyükbe lépő aktuális paraméterekkel hajtódik végre. A formális paraméterek a

SUBROUTINE szegmens leírásában csak a felhasználandó aktuális paraméterek szimbólumai.

Az eddigiekben nem részleteztük, hogy tulajdonképpen hol, milyen programrészben helyezkednek el az (5.15), (5.17) és (5.19) CALL utasítások. Az előző pontban szerzett ismereteink alapján látjuk, hogy (5.14) teljes programszegmens, első utasítása szegmensnyitó utasítás, utolsó utasítása pedig END utasítás. Az őt aktivizáló utasításnak nyilván valamelyik másik szegmensben kell elhelyezkednie. *Egy* SUBROUTINE *szegmenst mindig egy másik szegmensnek kell aktivizálnia.* Az aktivizálás eszköze a CALL utasítás. A CALL hatására — ideiglenesen — a vezérlés „átlép" a CALL utáni azonosítóval megjelölt szegmensbe, végrehajtja annak utasításait, majd „dolga végeztével" visszatér az őt aktivizáló szegmensbe, és ez utóbbinak utasításait a CALL-t követő FORTRAN utasítástól kezdve, folytatólag hajtja végre.

Honnan tudja meg a gép, hogy egy adott szegmensben teljesítette a feladatát, és visszatérhet az őt aktivizáló szegmensbe? Ennek a visszatérésnek az engedélyezésére van a RETURN utasítás. A RETURN a szegmens *logikai* végét jelenti, vagyis azt a pontot jelöli ki, ahol ki lehet lépni a SUBROUTINE szegmensből és vissza lehet térni az aktivizáló szegmensbe. Ilyen pont egy adott szegmensben esetleg több is lehet, és ennek megfelelően egy szegmensben több RETURN utasítás is előfordulhat. Általában *minden* SUBROUTINE *szegmensben kell legalább egy* RETURN *utasításnak lennie.* (Ugyanez érvényes a FUNCTION szegmensekre is.) Minthogy azonban egy szegmensnek csak egy *fizikai* vége van, tehát bármely szegmensben egy és csak egy END utasítás lehet.

Mivel a SUBROUTINE szegmenseket mindig egy másik szegmens aktivizálja (és ugyanez a helyzet a FUNCTION szegmensekkel is), nyilvánvaló, hogy ennek az aktivizálási folyamatnak valahol el kell indulnia; azaz minden programban kell lennie egy olyan szegmensnek, amelyet nem egy másik szegmens aktivizál, hanem például a gépkezelő, aki a vezérlőpultról elindítja a program végrehajtását. Szabatosabban fogalmazva, *minden végrehajtható* FORTRAN *programban kell lennie egy legelső végrehajtható utasításnak, ahonnan az egész számítási folyamat elindul.* Azt a programszegmenst, amely ezt az utasítást tartalmazza, a FORTRAN program *fő szegmensének* nevez-

zük. Az ASA FORTRAN nem határozza meg, hogy milyen módon kell ezt a szegmenst megjelölni, és a gépi reprezentációs nyelvek különféle speciális szegmensnyitó utasításokat vezetnek be erre a célra. Ilyenek a már említett PROGRAM, MASTER, MAIN stb. utasítások. A program első végrehajtható utasítása általában a fő szegmens első végrehajtható utasításával egyezik meg, de a gépi reprezentációs nyelvek esetleg ettől eltérően is meghatározhatják a program kezdetét. Minthogy a továbbiakban mindenképpen szükségünk lesz valamilyen megállapodásra a program első végrehajtható utasítására és fő szegmensére nézve, mint ahogyan a Negyedik fejezet 1. pontjában jeleztük, a jövőben a fő szegmens jelölésére a MASTER szegmensnyitó utasítást fogjuk használni, és a program kezdetének a MASTER szegmens első végrehajtható utasítását tekintjük. Hangsúlyozzuk azonban, hogy ez gépi reprezentációból (nevezetesen az ICT FORTRAN-ból) átcsempészett konvenció.

Eredeti példánkra, az (5.14) szubrutinra, valamint annak különböző aktivizálásaira visszatérve, egy szubrutint használó teljes FORTRAN programnak a következő sematikus szerkezete lehet:

```
MASTER KULACS
      :
CALL ORTOG (P, Q, R, S, ALFA)
      :
CALL ORTOG (A, B, C, D, THETA)                    (5.20)
      :
CALL ORTOG (A1 * * 2 – A2 * * 2, A1 + A2, R, S, BETA)
      :
END
      :
SUBROUTINE ORTOG (X, Y, U, V, FI)
      :
END
```

(5.20), mint látható, két szegmensből álló program, egy főprogrammal és egy szubrutinnal. Ennek természetesen nem kell szükségképpen így lennie; egy MASTER szegmensnek lehet több szubrutinja, és ezek a szubrutinok *egymást is* aktivizálhatják. Tehát magukban a SUBROUTINE szegmensekben is

lehetnek CALL utasítások, amelyek más SUBROUTINE szegmensek aktivizálására szólítanak fel. Mivel a RETURN utasítás végrehajtásakor a vezérlés mindig az aktivizáló szegmensbe tér vissza, ily módon szubrutin hívások tetszőleges hosszúságú láncolata jöhet létre, amelynek kiinduló pontja — és általában végpontja is — a MASTER szegmens. Azért *általában*, mert egy FORTRAN program végrehajtása megszakadhat egy, a MASTER-től különböző szegmensben is, amennyiben a gép ebben a szegmensben egy STOP utasítást hajt végre. Ez azonban csak kivételes esetekben szokásos, főként akkor, ha a SUBROUTINE szegmens által előírt valamilyen számítási feladat nem végezhető el, és a megállás éppen a feladat elvégezhetetlenségére kívánja felhívni a figyelmet (pl. negatív számból történő valós gyökvonás, szinguláris mátrix invertálása stb.). A tipikus programszervezés olyan, hogy a feladat eredményes befejezését nyugtázó STOP utasítás a MASTER szegmensben van elhelyezve.

2. Az (5.20)-nál valamivel bonyolultabb programsémát mutat be a következő példánk:

```
MASTER M
:
CALL S1 ...
:
CALL P1 ...
:
STOP
END
SUBROUTINE S1 (...)                    (5.21)
:
RETURN
END
SUBROUTINE P1 (...)
:
CALL P2 (...)
RETURN
```

```
END
SUBROUTINE P2 (...)
  :
IF(HIBA.LT.0) STOP
RETURN
END
```

Ebben a programban két különböző szubrutinhívási láncolatot láthatunk: a MASTER szegmens hívja az S1 és a P1 szubrutinokat, P1 pedig hívja P2-t. Ha a HIBA nevű változó negatív értéket vesz fel, a program végrehajtása megszakad P2-ben, ellenkező esetben a vezérlés P1-en keresztül visszatér a MASTER szegmensbe.

3. Legyen a feladat SUBROUTINE szegmens írása másodfokú algebrai egyenlet általános megoldására, figyelembe véve az elfajult esetek, valamint a komplex gyökök lehetőségét is. Jelölje a megoldandó másodfokú egyenletet

$$ax^2+bx+c = 0 \tag{5.22}$$

ahol az a, b és c együtthatók tetszőleges valós számok lehetnek. Az $a=b=0$ eset teljesen elfajult, ekkor az egyenlet megoldhatatlan. Ebben az esetben hajtsunk végre egy STOP utasítást! Az $a=0$, de $b\neq 0$ esetben az egyenlet lineárissá fajul; ekkor a két gyököt tekintsük megegyezőnek, és ezek $x_{12} = -c/b$-vel egyenlők. Végül, ha $a\neq 0$, akkor válasszuk szét a valós és komplex gyökök esetét.

Eljárásunk formális paraméterei a következők: jelölje A, B és C az együtthatókat, X1R, X1I, X2R és X2I pedig rendre az x_1, ill. x_2 gyök valós és képzetes részét. Vagyis A, B és C „bemenő" paraméterek, a többi négy pedig „kimenő" — azaz eredményt adó — paraméter. Szubrutin szegmensük a következő:

```
SUBROUTINE QVDSLV(A, B, C, X1R, X1I, X2R, X2I)
IF(A.EQ.0.0.AND.B.EQ.0.0) STOP 77
IF(A.EQ.0.0) GO TO 1
DISKR = B**2-4.*A*C
```

```
      ROOT=SQRT(ABS(DISKR))
      IF(DISKR) 2, 3, 3
    3 X1I=0
      X2I=0
      X1R = (-B+ROOT)/(2.0*A)
      X2R = (-B-ROOT)/(2.*A)
      IF(ABS(X1R)-ABS(X2R)) 4, 4, 5
    4 X1R=C/(X2R*A)                                   (5.23)
      RETURN
    5 X2R=C/(X1R*A)
      RETURN
    2 X1R = -B/(2.*A)
      X2R=X1R
      X1I = ROOT/(2.*A)
      X2I = -X1I
      RETURN
    1 X1I=0
      X2I=0
      X1R = -C/B
      X2R=X1R
      RETURN
      END
```

Az (5.23) szubrutinnak a különböző aleseteknek megfelelően négy visszatérési pontja van. Két valós gyök esetén két további alesetet választottunk szét aszerint, hogy a gyökök közül melyiknek nagyobb az abszolút értéke. A megoldóképletből csak a nagyobb abszolút értékű valós gyököt célszerű számítani,

a kisebb kiszámításához ajánlatosabb a gyökök és együtthatók összefüggésének felhasználása. Ily módon elkerülhetjük a kisebbik abszolút értékű gyök kiszámításakor a kivonási jegyvesztesé get.

Alkalmazzuk most ezt az eljárást a következő transzcendens egyenletrendszer megoldására:

$$\begin{aligned} px^2+x\sqrt{1+p}-1.1046 &= 0 \\ (x^2+1)p^2+\sqrt{1-x^2}p-0.73 &= 0 \end{aligned} \tag{5.24}$$

A rendszert x-re és p-re kell megoldani. Mindkét egyenlet valamelyik változójában másodfokú, és ez a körülmény azt sugalmazza, hogy próbálkozzunk iterációval: oldjuk meg először az első egyenletet valamilyen p_0 kezdőértékkel x-re; helyettesítsük a kapott értéket a második egyenletbe és oldjuk meg ezt p-re; ezt az értéket ismét az elsőbe helyettesítve, állítsunk elő egy újabb x közelítő értéket, majd ebből egy újabb p-t s í. t. Az egyenlettel szerencsénk van, mert ez az iteráció alkalmas p_0-ból kiindulva valóban konvergál. Nem magától értetődő, de írjuk úgy a programot, hogy a következő iterált előállításához mindig az előző első gyökét (vagyis a pozitív négyzetgyökkel számítottat) használjuk fel. Legyen a megkövetelt abszolút pontosság 10^{-6}. A feladat megoldásának fő szegmense a következő lehet:

```
       MASTER MEP
       REAL IMAG1, IMAG2
       DATA IMAG1, IMAG2/2*0.0/
       READ (...) P
       PE=P
       XE=P
       NITER=0
     1 CALL QVDSLV(P, SQRT(1+P), -1.1046, X, IMAG1,
     X RE2, IMAG2)
       IF(IMAG1.NE.0.0) STOP 01
```

```
      CALL QVDSLV(X**2+1., SQRT(1.-X**2),
     X-0.73, P, IMAG1, RE2, IMAG2)
      IF(IMAG1.NE.0.0) STOP 01
      NITER = NITER+1
      IF(ABS(PE-P).LT.1E-6.AND.ABS(XE-X).LT.1E-6)
     XGO TO 2
      IF(NITER.GT.25) STOP 01
      GO TO 1
    2 WRITE (...) NITER, X, P
      STOP 02
      ...
      END                                       (5.25)
```

A programban a READ és WRITE utasítások a kezdőérték beolvastatásának, illetve az eredmény kiíratásának utasításait szimbolizálják. Ezeknek az utasításoknak a használatával a Hatodik fejezetben ismerkedünk majd meg. A READ és WRITE utasításokhoz tartozik egy-egy, ugyancsak a Hatodik fejezetben megismerendő FORMAT utasítás is, melyek helyét a program végén kipontoztuk.

A programban előforduló NITER változó az iterációs lépéseket számlálja. Ha ez meghaladja a 25-öt, a program megáll a 01 jelzésű STOP utasítások egyikén. Ez vészjelzés, a konvergencia nem elég gyors. Ugyanez adódik abban az esetben is, ha a másodfokú egyenletek megoldása közben komplex gyök fordul elő, ugyanis ez szintén az eljárás divergenciájához vezet. A képzetes részeket tartalmazó változók kezdőértékét a program elején álló DATA utasítás zérusra állítja be. A vészjelző megállások arra utalnak, hogy a beolvasott kezdőérték nem volt megfelelő. Az iteráció eredményes befejezését — az eredmények kinyomtatása után — a 02-vel megjelölt STOP utasítás jelzi. A főprogram CALL utasításai példát mutatnak arra az esetre, amikor az eljárás aktivizálásakor nem változót, hanem aritmetikai kifejezést adunk meg aktuális paraméterként.

4. Írjuk meg szubrutin szegmens formájában a mátrixszorzás már ismert algoritmusát! Legyenek az összeszorzandó mátrixok 10×10-esek!

A feladatnak megfelelő szubrutin szegmens a következő:

```
      SUBROUTINE MTXMLT (A, B, C)
      DIMENSION A(10,10), B(10,10), C(10,10)
      DO 1 I=1, 10
      DO 1 J=1, 10
      WORK=0.0                                    (5.26)
      DO 2 K=1, 10
    2 WORK = WORK+A(I, K)*B(K, J)
    1 C(I, J)=WORK
      RETURN
      END
```

(Itt az 1 és 2 címkék DO ciklusok záróutasításain vannak elhelyezve.) Az (5.26) szubrutin szegmens felhasználásával szoroztassuk össze az X és Y, valamint az U és V 10×10-es mátrixokat! Az eredmény legyen ugyancsak 10×10-es Z, ill. W mátrix. Írjuk be a programba az összeszorzandó mátrixok beolvasását és az eredmények kiíratását szimbolizáló utasításokat is!

A megoldás MASTER szegmense a következő lehet:

```
      MASTER FOPROG
      DIMENSION X(10,10), Y(10,10), Z(10,10),
     X U(10,10), V(10,10), W(10,10)
      READ (...) X, Y, U, V                       (5.27)
      CALL MTXMLT(X, Y, Z)
      CALL MTXMLT(U, V, W)
```

```
WRITE (...) Z, W
STOP
...
END
```

A MASTER szegmens az összeszorzandó mátrixok beolvasásával kezdődik; ezt követi a két CALL utasítás, amely kétszer aktivizálja — különböző aktuális paraméterekkel — az MTXMLT szubrutint. Az eredménymátrixok kinyomtatása után a gép megáll. A szegmens végén a FORMAT utasítások helyét pontoztuk ki.

Ez a példa talán valamennyire érzékelteti a programszegmentálás előnyeit. Ha ugyanis a példánkban szereplő mátrixokat a szegmentálás lehetőségének kihasználása nélkül kívántuk volna összeszoroztatni, programunk terjedelme jócskán megnőtt volna, mert vagy a mátrixszorzás háromszoros ciklusát kellett volna kétszer leírnunk, vagy egy kétszeres ciklust kellett volna beiktatnunk az összeszorzandó mátrixok áthelyezésére adott azonosítóval ellátott tömbökbe (ez utóbbi megoldás gépidőveszteséggel is járna). A mátrixszorzási algoritmusnak szubrutin formában való megadása automatikusan megoldja a feladatban fellépő „átjelölési" problémát. A későbbiekben azt is látjuk majd, hogy a szegmentálás felhasználásával hogyan lehet „átdimenzionálni" a programban szereplő tömböket.

Most — miután már kialakult valamelyes szemléletes képünk a több szegmensből álló programokról és ezek használatáról — áttérünk a paraméter átadás—átvétel szabályainak szabatos tárgyalására. A cél annak tisztázása, hogy milyenek lehetnek egy eljárás formális, ill. aktuális paraméterei, milyen kapcsolatban vannak ezek egymással, és hogyan kell őket megadni.

Annyit már a 2. pontban is mondottunk a formális paraméterekről, hogy tetszőleges azonosítók lehetnek. Ez igaz is, a kérdés azonban az, hogy milyen programozási objektumoknak lehetnek az azonosítói? E kérdésre a válasz a következő: *a formális paraméterek azonosítói jelölhetnek skaláris változót, tömböt vagy másik szegmenst.* A formális paraméterekhez hozzárendelt aktuális paraméterek megadási szabályai a követ-

kezők (ezeket részben már az előzőkben megismertük, most azonban összefoglalva közöljük):

1. A formális, és a nekik megfelelő aktuális paraméterek számának és sorrendjének meg kell egyeznie.

2. Ha a szegmens valamelyik formális paramétere *skaláris változó*, akkor a neki megfelelő aktuális paraméternek vele megegyező típusú *skaláris* vagy *indexes változónak*, vagy *kifejezésnek* kell lennie. A formális paraméter típusát szükség esetén megfelelő típusdeklaráló utasításokkal deklarálhatjuk (lásd a Negyedik fejezet 3. pontját). Deklaráció hiányában a típus az azonosítókra tett általános konvenciók szerint határozódik meg (lásd a Második fejezet 3. pontját).

3. A szubrutin aktivizálásának pillanatában az aktuális paraméterként megadott *kifejezés* értékének meghatározottnak kell lennie, és ennek az értéknek a megváltoztatása a szegmens végrehajtása közben *nem megengedett*. Hasonlóan, ha az aktuális paraméter *indexes változó*, akkor az *indexkifejezésnek* a behívás pillanatában meghatározottnak kell lennie.

4. Ha a szegmens valamelyik formális paramétere *tömb*, akkor ennek méreteit — esetleg dinamikusan, lásd a 4. pontot — a szegmensben deklarálni kell. Az aktuális paraméter ebben az esetben tömbazonosító vagy indexes változó lehet. Ha az aktuális paraméter *tömb*, akkor elemeinek száma legalább akkora kell hogy legyen, mint a formális paraméter tömb elemeinek száma. Ha az aktuális paraméter *indexes változó*, akkor a formális paraméter tömb elemeinek számára a következő korlátozásnak kell teljesülnie:

$$f \leqq a - k(\mathsf{I}, \mathsf{J}, \mathsf{K}) + 1 \tag{5.28}$$

ahol f a formális paraméter tömb elemeinek száma, a az aktuális paraméter tömb elemeinek száma, és k az aktuális paraméter tömb címfüggvényének (lásd a Negyedik fejezet 3*d* pontját) helyettesítési értéke az I, J, K helyeken (ha az aktuális indexes változónak nem három, hanem két, ill. egy indexe van, akkor (5.28)-ban értelemszerűen $k(\mathsf{I}, \mathsf{J})$ ill. $k(\mathsf{I})$ írandó). Nem követeljük meg, hogy az aktuális és a formális tömb dimenziószáma megegyezzék.

5. Ha a szegmens valamelyik formális paramétere *szegmensazonosító*, akkor a megfelelő aktuális paraméternek is szegmensazonosítónak kell lennie. Az aktuális paraméterként

megadott szegmensazonosítót a behívó (nem a behívott!) szegmensben EXTERNAL-ként kell deklarálni (lásd 7. pont).

6. Aktuális paraméterként a fentieken kívül megadható *Hollerith konstans* is, ha a megfelelő formális paraméter skaláris változó vagy tömb. (Az ASA FORTRAN keretein belül ezt a lehetőséget legfeljebb szövegek kiíratására lehet kihasználni, minthogy karaktereken operáló utasítások a hivatkozási nyelvbe nincsenek beépítve.)

7. Különböző formális paramétereknek csak abban az esetben feleltethető meg ugyanaz az aktuális paraméter, ha az előbbiek értékét a szubrutin végrehajtása során semmilyen módon nem változtatjuk meg (röviden szólva: csak a bemenő paramétereket szabad azonosnak előírni).

8. A formális paraméterek COMMON, EQUIVALENCE és DATA utasításokban nem fordulhatnak elő.

9. Egyetlen szubrutin sem aktivizálható újra mindaddig, amíg a lezáró RETURN utasítás végrehajtására sor nem került (azaz tilos az eljárások bármilyen rekurzív aktivizálása).

A fenti szabályok némelyikéhez szükséges bizonyos magyarázatot fűznünk.

A 3. szabály első részének mondanivalóját abban jelölhetjük meg, hogy a FORTRAN nyelvben az aktuális paraméter kifejezések átvétele *érték szerint* történik. Ez azt jelenti, hogy a gép *előbb kiszámítja* a kifejezés értékét, majd ezután tér a szubrutin végrehajtására, és a formális paraméterre való hivatkozások a végrehajtás folyamán erre az értékre történő hivatkozásokba mennek át (lásd az (5.25) példában a QVDSLV szubrutin különböző behívásait). A hangsúly itt azon van, hogy az eljárás aktivizálása az aktuális paraméter kifejezések kiértékelésével kezdődik, és a gép ezt a műveletet a szubrutin végrehajtása közben többször már nem ismétli meg.

Az olvasónak úgy tűnhet, hogy itt érdemén felüli nyomatékkal hangsúlyozunk egy trivialitást, hiszen mi sem természetesebb, mint az, hogy az aktuális paramétert *egyszer* előre kiszámítjuk, majd a továbbiakban ezzel az értékkel dolgozunk. A helyzet azonban az, hogy nem is túlzottan nagy ügyeskedéssel lehet olyan FORTRAN programot írni, amelyben a behívott szegmens menet közben megváltoztatja valamelyik aktuális paraméter kifejezésének értékét és ebben az esetben nem közömbös, hogy az ezt követő hivatkozásoknál a kifejezésnek az *eredeti* (vagyis a behíváskor viselt), vagy pedig a *módosított* (vagyis a szubrutin végrehajtása során előállított) értéke veendő figyelembe.

Ezt a kérdést egyébként az ALGOL 60 nyelv ismerői szokták élére állítani. Ott ugyanis megkülönböztetünk ún. érték szerinti és név szerinti paraméter átvételt. A kérdés az, hogy a kettő közül melyiknek felel meg a FORTRAN nyelv paraméter átvételi rendszere?

A válasz egyiknek sem. Az ALGOL nyelv terminológiájával fogalmazva, a pontos helyzet a következő:

1. A kifejezések átvétele érték szerint történik, azonban a FORTRAN explicite megtiltja az olyan formális változó értékének megváltoztatását, amelyhez aktuális paraméterként kifejezés van hozzárendelve.

2. A tömbök átvétele lényegileg név szerint történik, azzal az eltéréssel, hogy a méreteket a behívott szegmensben is deklarálni kell, és a formális és aktuális méretek között a 4. számú kompatibilitási szabálynak kell fennállnia.

3. Ha skaláris változó a formális paraméter, melynek aktuális paraméterként skaláris vagy indexes változó felel meg, akkor ennek átvétele lényegileg ugyancsak név szerint történik, azzal, hogy az aktuális paraméterként megadott indexes változók indexeinek — az ALGOL-tól eltérően — az eljárás behívásakor szükségképpen meghatározottnak kell lenniük.

4. Annak az esetnek, amikor a formális paraméter tömb, az aktuális paraméter pedig indexes változó, az ALGOL-ban egyáltalában nincs megfelelője, valamint annak a lehetőségnek sem, hogy az aktuális és a formális tömb méretének (dimenzióinak) száma eltérhet.

A negyedik paraméter átvételi szabály szerint formális paraméter tömbnek megfeleltethetünk aktuális paraméterként nemcsak tömböt, hanem indexes változót is. A következőkben ezt fejtjük ki kissé részletesebben.

Az említett lehetőség módot ad a formális paraméter tömb „kezdetének" a kijelölésére. Az ilyen paraméter megadásakor ugyanis a fordítóprogram az aktuálisként megadott tömbelemet azonosítja a formális paraméter tömb első elemével, „ettől az elemtől kezdve" számítja a formális paramétert.

Példa

Tegyük fel, hogy rendelkezésünkre áll egy FORTRAN szubrutin, amely 20 elemű vektorok skaláris szorzatának kiszámítására alkalmas. A feladat 5 ilyen skaláris szorzás elvégzése. Az említett feladatot természetesen megoldhatjuk „elemi" eszközökkel is, azonban a „tömb—tömbelem" megfeleltetés a programot lényegesen egyszerűsítheti. Álljon itt mindkét lehetséges megoldás!

Mindkettőben szükség van a skaláris szorzást végző szubrutinra, mely a következő alakú lehet:

```
  SUBROUTINE SKLMLT(U, V, SZORZAT)
  DIMENSION U(20), V(20)
  SZORZAT=0.0
  DO 1 I=1, 20                                  (5.29)
1 SZORZAT = SZORZAT+U(I)*V(I)
  RETURN
  END
```

A feladat megoldása elemi eszközökkel (vö. (5.27)):

```
  MASTER TRIVI
  DIMENSION A1(20), B1(20), A2(20), B2(20),
 XA3(20), B3(20), A4(20), B4(20), A5(20), B5(20)
  READ (...) A1, B1, A2, B2, A3, B3, A4, B4, A5, B5
  CALL SKLMLT (A1, B1, RES1)
  CALL SKLMLT (A2, B2, RES2)                    (5.30)
  CALL SKLMLT (A3, B3, RES3)
  CALL SKLMLT (A4, B4, RES4)
  CALL SKLMLT (A5, B5, RES5)
  WRITE (...) RES1, RES2, RES3, RES4, RES5
  STOP
  ...
  END
```

A program elején álló READ utasítás a számításban résztvevő tíz vektor beolvastatását szimbolizálja. Ezt öt szubrutinhívó utasítás követi, amelyek különböző paraméterek esetén kiszámíttatják az egyes skalárszorzatokat. A program az eredmények kiíratásával végződik.

Lássuk most, mennyiben lehet segítségünkre a feladat megoldásában, ha a tömbparamétert indexes változó aktuális paraméterrel helyettesítjük. A módosított MASTER szegmens a következő:

```
  MASTER AGYAF
  DIMENSION A(100), B(100), RES(5)
  READ (...) A, B
  DO 1 I=1, 5                                        (5.31)
1 CALL SKLMLT (A(20*I-19), B(20*I-19), RES(I))
  WRITE (...) RES
  STOP
  ...
  END
```

Az (5.31) programváltozatban — az előzővel ellentétben — mindössze három vektort deklarálunk, amelyek közül az első kettő a skalárisan összeszorzandó vektorok elemeit tartalmazza. Ezeknek a vektoroknak egy-egy 20 elemből álló „szakasza" alkot egy vektortényezőt, és ezeken egy DO ciklusban haladunk végig. A szorzásban résztvevő 20 elemű vektorok — a méretet az SKLMLT szubrutin deklarációs része határozza meg — kezdete az első ciklusban A(1) és B(1), a másodikban A(21) és B(21), a harmadikban A(41) és B(41) s. í t. A DO ciklus lehetővé tette, hogy az eredményeket is egy vektornak az elemeiben helyezzük el.

A formális paraméter tömb elemeire kikötött (5.28) korlátozásnak, úgy hisszük, most már világos az értelme: azt fejezi ki, hogy a formális paraméter tömb akkor sem nyúlhat túl az aktuális tömbön, ha a paraméter hozzárendelés folytán a formális tömböt nem az aktuális tömb első eleménél „kezdjük meg".

A tömb—tömbelem hozzárendelésnek hasznos gyakorlati alkalmazásai vannak, ha a dinamikus indexhatárok lehetőségével kombinálva alkalmazzuk (lásd a 4. pont (5.51) példáját).

A 7. szabály arra az esetre rendelkezik, ha egy szubrutinnak két aktuális paramétere azonos. Ennek az a következ-

ménye, hogy két, formálisan különböző paraméter a behívás következtében egymással azonossá válik. Ez megzavarhatná az eljárás működését.

A 7. szabály rendelkezéseinek megfelelően, a

CALL MTXMLT(P, P, R) (5.32)

CALL MTXMLT(P, R, P) (5.33)

szubrutinhívó utasítások közül az első korrekt, mivel az (5.26) programrészlet szerint az első két paraméternek megfelelő tömb elemeinek értéke a szubrutin végrehajtása során nem változik meg, de a második nem korrekt, mert a szubrutin harmadik paraméterének megfelelő tömb elemei új értéket vesznek fel.

b) FUNCTION szegmensek és alkalmazásaik

Az előző alpontban megismerkedtünk a SUBROUTINE szegmensek felírási és hívási szabályaival. Az ott megismert szabályok legnagyobb része a FUNCTION szegmensekre is érvényes, ezért utóbbiakkal már jóval rövidebben végezhetünk.

A FUNCTION és a SUBROUTINE szegmensek között a leglényegesebb különbség az, hogy a FUNCTION szegmensek azonosítójához a szegmens végrehajtása egy — szám-, vagy logikai — értéket rendel, míg a SUBROUTINE szegmensek esetén ilyen érték hozzárendelésére nem kerül sor. Ezenkívül a FUNCTION szegmensek aktivizálásának módja is eltér a SUBROUTINE szegmensekétől, amennyiben az előbbiek behívása nem CALL utasítással, hanem az aritmetikai utasításfüggvények (lásd a Negyedik fejezet 6. pontját) mintájára történik.

A függvények meghatározására szolgáló FUNCTION szegmensek nyitó utasításának formai szabályait már a 2. pontban megismertük (lásd (5.7) és (5.8)). A szegmensnyitó utasításon kívül van azonban a FUNCTION szegmenseknek még egy ismérvük, amely szorosan összefügg a FUNCTION szegmens által meghatározott érték meghatározásával:

Minden FUNCTION *szegmensben a szegmens azonosítójának legalább egyszer skaláris változó módjára elő kell fordulnia valamelyik végrehajtható utasításban, és ezen előfordulások között kell olyannak is lennie, amely a szegmens nevével jelzett változó értékét meghatározza. A szegmens végrehajtásának bármilyen*

lefutása mellett legalább egy ilyen értékmeghatározásra sor kell hogy kerüljön. A szegmens nevével azonosított skaláris változó azon értékét, amelyet a szegmens RETURN *utasításának végrehajtása pillanatában visel, a szegmens által generált függvényértéknek nevezzük.*

Példák

1. Írjunk FUNCTION szegmenst a sh x függvény kiszámítására! A számítás során felhasználhatjuk a függvény ismert Taylor-sorfejtését, amely szerint

$$\operatorname{sh} x = x+\frac{x^3}{3!}+\frac{x^5}{5!}\dots+\frac{x^{2n-1}}{(2n-1)!}+\dots \tag{5.34}$$

A sor minden valós x-re konvergens, bár nagy abszolút értékű argumentumok esetén egyre messzebbre kell elmennünk, hogy elegendően pontos közelítéshez jussunk.

A bemutatandó eljárás első változatának egyetlen paramétere van, a függvény argumentuma. A FORTRAN program a következő:

```
  FUNCTION SH(X)
  Z = X**2
  SH=X
  WORK=X
  K=3
1 WORK = Z/FLOAT(K*(K-1))*WORK
2 SH = SH+WORK
3 IF(WORK/SH.LT.1E-7) RETURN
  K = K+2
  GO TO 1
  END
```

(5.35)

Hasonló algoritmussal találkoztunk már a Harmadik fejezetben is (lásd (3.40)); itt a lényeg a programrészletnek FUNCTION

szegmenssé való kiegészítése. Figyeljük meg a programban az SH szegmensazonosítónak a szerepét! A szegmens eleget tesz az ASA FORTRAN ama szabályának, mely szerint a szegmensazonosítónak a szegmensben skaláris változó szerepében legalább egyszer elő kell fordulnia: itt négy ilyen előfordulás van, melyek közül kettő „aktív", azaz meghatározza, ill. módosítja SH értékét. Az is teljesül, hogy a RETURN végrehajtása pillanatában a szegmens azonosítójának értékkel kell bírnia, mert a RETURN eléréséig minimálisan két (de esetleg jóval több) értékadás történt az SH változóra. A program az (5.34) sor tagjait addig generálja és összegezi, amíg a járulékos tag — a WORK programbeli változó — értéke a sor többi tagjához képest elhanyagolhatóan kicsinnyé nem válik.

Az ALGOL nyelv ismerői feltehetőleg észrevették, hogy itt ismét szembetűnő eltérés van a két nyelv szabályai között, amennyiben a FORTRAN megengedi a szegmens azonosítójának „passzív" előfordulását a szegmens végrehajtható utasításaiban, az ALGOL viszont kifejezetten megtiltja, illetőleg az eljárás rekurzív aktivizálásaként értelmezi. Ha az (5.35) programot ALGOL-ba kívánnánk átírni, akkor az SH azonosítót a 2-es és a 3-as címkével megjelölt utasításokban szereplő aritmetikai kifejezésből valamilyen programozási fogás alkalmazásával el kellene tüntetnünk.

A végrehajtható utasításokkal ellentétben, deklaratív utasításokban nem fordulhat elő a FUNCTION szegmens azonosítója: *a* FUNCTION *utasítást kivéve, a szegmens azonosítója semmilyen deklaritív utasításban nem szerepelhet.*

Bizonyos gépi reprezentációs nyelvek — egyetlen kivételként — megengedik a FUNCTION szegmens nevének típusdeklaráló utasításban való megjelenését, mint pl.:

```
FUNCTION EGESZF(I)
                                  (5.36)
INTEGER EGESZF
```

Hangsúlyozzuk, hogy az ilyen deklaráció ellenkezik az ASA FORTRAN szabályaival. ASA FORTRAN-ban (5.36) helyesen így fest:

```
INTEGER FUNCTION EGESZF(I)        (5.37)
```

(5.37)-ben az INTEGER típusdeklaráló szó az előállítandó függvényérték típusát jelöli ki. Ilyen típusdeklaráció hiányában a

típus az azonosító első betűje alapján, a szokott módon, az automatikus típusdeklaráció szerint határozódik meg.

Akárcsak a SUBROUTINE szegmenseknek, a függvényeknek is tetszőleges számú formális paraméterük lehet, és ily módon FUNCTION szegmens segítségével akárhány változós függvények is előállíthatók. Az (5.35) eljárás például tartalmazhatja formális paraméterként az elérendő relatív pontosságot. A megfelelő módosított változat (csak az eltérő utasításokat írtuk ki):

```
  |FUNCTION SH(X,EPS)
  |...
 3|IF(WORK/SH.LT.EPS) RETURN                    (5.38)
  |...
```

A FUNCTION szegmensek aktivizálására vonatkozóan már említettük, hogy azokat a standard függvényekhez, vagy az aritmetikai utasítás függvényekhez hasonlóan, azonosítójuk és aktuális paramétereik megadásával hívhatjuk be. Akárcsak a már ismert függvényfajták, a FUNCTION szegmensek azonosítója is beírható aritmetikai, ill. logikai kifejezésekbe, és azonosítójuk ilyen előfordulása a függvényeljárás aktivizálását jelenti. Például:

```
V = A+SH(B**2-COS(RO))                          (5.39)
IF(FVERT(X+DELTAX).GT.BETA) GO TO 2             (5.40)
IF(LOGF(BOLEAN)) LOGIC=.TRUE.                   (5.41)
```

(5.39) az (5.35) FUNCTION szegmens egy lehetséges aktivizálását mutatja. Az (5.40) példa egy FVERT nevű FUNCTION szegmens aktivizálását illusztrálja, végül (5.41) egy logikai értéket előállító függvény használatára ad példát. Utóbbi szegmensnyitó és első deklaratív utasítása a következő lehet:

```
 |LOGICAL FUNCTION LOGF(ARG)
 |LOGICAL ARG                                   (5.42)
 |...
```

A paraméterátvételre vonatkozó szabályok (lásd 142. oldal) a FUNCTION szegmensekre pontosan ugyanúgy érvényesek,

mint a SUBROUTINE szegmensekre, így azokat nem ismételjük meg. Talán a 9. szabályról érdemes annyit megjegyeznünk, hogy a SH(SH(X)) típusú aktivizálások annak nem mondanak ellent, mert a „belső" aktivizálás ebben az esetben előbb befejeződik (végrehajtódik a „belső" RETURN), és csak ezután kezdődik meg a „külső" aktivizálás. Tehát a függvényhivatkozások ilyen módon egymásba skatulyázhatók.

Van az ASA FORTRAN nyelvnek néhány olyan járulékos szabálya, amely csak a FUNCTION szegmensekre vonatkozik. A következőkben ezekkel foglalkozunk.

1. A SUBROUTINE szegmensekhez hasonlóan, a FUNCTION szegmensek is újra definiálhatják bizonyos formális paramétereik értékét, és ezáltal a függvény „főértékén" (vagyis a szegmens azonosítójához rendelt értéken) kívül „mellékértékeket" is előállíthatnak. (Példa lehet (5.35)-nek egy olyan módosítása, amelyben a sor tagjait számláló K változót is felvesszük a formális paraméterek listájára. Az eljárás ebben az esetben járulékos értékként a sorfejtésben felhasznált legmagasabbfokú tag hatványkitevőjét állítja elő.)

2. Megköveteljük, hogy ha valamely aritmetikai vagy logikai kifejezés kiértékelése közben ugyanazt a FUNCTION szegmenst kétszer teljesen azonos paraméterek mellett aktivizáljuk, akkor a két behívás — az aktivizálások sorrendjétől függetlenül — ugyanazt a függvényértéket állítsa elő.

Ezen utóbbi, második szabály elvezetett minket az ún. *mellékhatások* problémájához.

Mellékhatáson azt értjük, hogy valamely eljárás megváltoztatja az őt behívó szegmens bizonyos változóinak értékét, és ezáltal egyes kifejezések értéke a kiszámítás sorrendjétől függővé válik. Ilyen példát igen könnyű felírni:

```
INTEGER FUNCTION CSEL (I,K)
CSEL = I+K
K = I+1                                   (5.43)
RETURN
END
```

Aktivizáljuk az (5.43) függvényt a következő utasításokkal:

$$\begin{array}{l} J=2 \\ N = CSEL(2,J)+J \end{array} \tag{5.44}$$

A CSEL függvény kiértékelése J értékét 3-ra növeli. Ezért attól függően, hogy az összeg kiértékelését J értékének megjegyzésével, vagy a CSEL függvény kiszámításával kezdjük, N a 6, vagy a 7 értéket veszi fel. A kifejezés persze nem korrekt az ASA FORTRAN ama szabálya miatt, amely szerint az elemi matematika azonosságainak (kommutatív, asszociatív és disztributív törvény) — az esetleges kerekítési hibák hatásától eltekintve — a FORTRAN kifejezésekre is teljesülniük kell (lásd a Második fejezet 5. pontját).

Az iménti 2. szabály bizonyos formális védelmet nyújt a mellékhatások ellen, kikötve, hogy a kétszeri aktivizálásnak — azonos paraméterek mellett — azonos eredményt kell adnia. Az (5.43) függvény ezt a kikötést éppen nem teljesíti, mert pl. az

$$N = CSEL(2,J)+CSEL(2,J) \tag{5.45}$$

értékadó utasításban a függvény eltérő értékeket szolgáltat az első és második aktivizáláskor. A szabály által nyújtott védelem azonban csak részleges, mert egy ebből a szempontból kifogástalan eljárásnak is lehet mellékhatása. Még az sem nyújtana teljes biztonságot, ha megtiltanánk a FUNCTION szegmensekben a „mellékértékek" generálását (a CSEL függvény második paramétere ilyen mellékérték, és az adott esetben ez okozza a bajt), mert a közös adatmezők (lásd az Ötödik fejezet 5. pontját) nyújtotta lehetőségekkel visszaélve, még mindig lehetne mellékhatásos függvényeket felírni. Az egyértelmű megoldást — amint az ALGOL nyelvben így is van — csak a szigorú balról jobbra szabály biztosítja, ez azonban gátja a kifejezések kiszámításában az optimalizálási lehetőségek kihasználásának.

Befejezésül felhívjuk a figyelmet arra a tényre, hogy az aritmetikai utasítás függvények és a FUNCTION szegmensek között nem az a leglényegesebb különbség, hogy az előbbiek csak egyetlen aritmetikai kifejezéssel megoldható formulák kiszá-

mítására használhatók, míg a FUNCTION szegmensek bonyolultabb számítási eljárásokkal megadható függvénykapcsolatok kiszámíttatására is alkalmasak. A legfontosabb eltérés az, hogy *az aritmetikai utasítás függvények azonosítója nem szegmensközi azonosító*, így az utasítás függvények csak abban a szegmensben használhatók, amelyben deklaráljuk őket. A FUNCTION szegmensek viszont a program bármely szegmenséből aktivizálhatók.

FUNCTION szegmensek alkalmazására példákat e fejezet további pontjaiban is látunk majd.

4. Dinamikus indexhatárok

Az eddig megismert, tömbparamétereken operáló eljárások (pl. (5.26) és (5.29)) rugalmas felhasználásának az az akadálya, hogy a szubrutin megírásakor előre lerögzítettük a formális paraméter tömb méreteit. Ez nyilván nem kedvező, mert egy olyan szubrutint, amelyet 10×10-es tömbök szorzására írtunk meg, nem használhatunk fel, ha a feladat történetesen 20×20-as méretet írna elő. Már az 1. pontban felmerült a tömbméret rugalmas megválasztásának problémája. A jelen pontban ennek módozataival ismerkedünk meg.

Az ASA FORTRAN lehetőséget ad arra, hogy a SUBROUTINE és FUNCTION szegmensekben deklarált formális paraméter tömbök méretét *változóként* definiáljuk. Ilyen esetben a DIMENSION (vagy típusdeklaráló) utasításban az indexhatárokat jelző számok helyett *index nélküli*, *egész változók* is állhatnak, melyeknek maguknak is formális paramétereknek kell lenniük, a szegmens aktivizálásakor aktuális értékkel kell rendelkezniük, és értéküket a szegmens végrehajtása során nem szabad megváltoztatniuk. A méretet definiáló változók aktuális értékeinek olyannak kell lenniük, hogy — akárcsak skaláris indexhatárok esetén — a formális tömb beleférjen az aktuális tömbbe. Az ily módon deklarált formális tömbről azt mondjuk, hogy *dinamikus indexhatárai* vannak.

Példa

Az (5.26) szubrutin átírása dinamikus indexhatárok alkalmazásával:

```
      SUBROUTINE MTXMLT (A, B, C, L, M, N)
      DIMENSION A(L,M), B(M,N), C(L,N)
      DO 1 I=1, L
      DO 1 J=1, N
      WORK=0.0
      DO 2 K=1, M                                    (5.46)
    2 WORK = WORK + A(I,K) * B(K,J)
    1 C(I,J) = WORK
      RETURN
      END
```

Az eljárást mindjárt úgy írtuk át, hogy alkalmas legyen tetszőleges — nem okvetlenül négyzetes — mátrixok összeszorzására. Ha az összeszorzandó mátrixok pl. 10×5, ill. 5×20 méretűek, akkor a megfelelő szubrutinhívó utasítás

```
CALL MTXMLT(X, Y, Z, 10, 5, 20)        (5.47)
```

lehet. Az eredménymátrixnak 10×20-asnak kell lennie.

A dinamikus indexhatárok alkalmazásában éppen az az eset a tipikus, amikor az aktuális és a formális tömb mérete nem egyezik meg. A programokat ugyanis rendszerint úgy írják, hogy a MASTER szegmensben a tömbök méretét az előfordulható legnagyobbra deklarálják, míg a tényleges méretek ezt általában nem érik el. Tegyük fel pl., hogy olyan feladatunk van, amelyben változó méretű mátrixokat kell összeszoroztatnunk, és az előfordulható maximális méret 20×20-as, az aktuális méretek azonban ennél kisebbek is lehetnek. Ebben az esetben a MASTER szegmens szervezése a következő:

```
      MASTER PLD
      DIMENSION X(400), Y(400), Z(400)
      READ (...) L, M, N
      LIMIT = L * M
```

```
      DO 10 I=1, LIMIT
      ...
      X(I)=...
10    CONTINUE
      LIMIT = M*N
      DO 20 I=1, LIMIT
      ...
      Y(I)=...
20    CONTINUE
      CALL MTXMLT (X, Y, Z, L, M, N)
      ...
      STOP
      END
```
(5.48)

Ebben a példában a MASTER szegmens tömbjeit egyméretűként deklaráljuk, maximálisan 400 elemmel, ami éppen egy 20×20-as mátrix elemszámának felel meg. A két DO ciklus az egyes tömbök elemeinek előállítását szimbolizálja. A tömbelemek előállítása után kerül sor a mátrixszorzásra. Ez egyben példa arra is, hogy kétindexes formális paraméternek egyindexes aktuális paramétert feleltetünk meg. Ez nem hiba, a megszorítások csakis az elemszámra vonatkoznak, a méretek (dimenziók) számát nem érintik.

Felmerül a kérdés, hogy miért nem lehet, vagy miért nem célszerű ilyen esetben az aktuális paramétert is annyi indexesként deklarálni, mint ahány indexes a formális paraméter? Ennek oka, hogy ha az aktuális és a formális paraméter indexhatárai különbözők — a jelen példában is ez lenne a helyzet —, akkor *a formális és az aktuális paraméter tömb címfüggvénye különböző*. Ezt azonnal belátjuk, ha meggondoljuk, hogy az aktuális paraméter címfüggvénye — 20×20-as méretet feltételezve —

$$f(I,J) = I-1+20(J-1)+X \tag{5.49}$$

míg az (5.47) szubrutinhívás számadataival, az (5.46) eljárás A formális paraméteréé

$$g(\mathsf{I},\mathsf{J}) = \mathsf{I}-1+10(\mathsf{J}-1)+\mathsf{X} \tag{5.50}$$

(Az additív állandó itt az aktuális—formális paraméter megfeleltetés miatt X és nem A!)

A lényeg az, hogy a paraméter megfeleltetés csak a *tömbkezdetek egyeztetését* jelenti, de mindkét tömb megőrzi a saját címfüggvényét. Ennek nyilvánvaló következménye, hogy ha a MASTER szegmensben értéket adunk mondjuk az X tömb (3, 2) indexű elemének, akkor ez az MTXMLT szegmensben nem az A(3,2) tömbelemre, hanem ennek a tömbnek egy másik (történetesen (3,3) indexű) elemére jelent értékadást; az X tömb (3,4) indexű elemének pedig az A tömb egyik eleme sem felel meg. Jól vigyázzunk tehát! Ha az aktuális és formális tömb *méretszáma* megegyezik, de *indexhatáraik* különböznek, akkor az azonos számindexekkel megjelölt elemek az aktuális és a formális paraméterben *nem ugyanazon a helyen vannak a memóriában*, és a megfelelő tömbelemek indexeit a következési függvényeknek esetenkénti átszámítása alapján kell meghatározni. Ezt legegyszerűbben az (5.48) példán bemutatott módon kerülhetjük el.*

Még mindig az iménti példánál maradva, feltehetnénk a kérdést, nem lehetne-e azt memória szempontjából gazdaságosabban megoldani? A program ugyanis a benne szereplő mátrixok számára lefoglal 400+400+400 = 1200 tömbelem helyet, amit általában igen rosszul használ ki. A deklarációk ily módon történő megadása ugyanis nem teszi lehetővé pl. egy 25×25-ös mátrixnak egy 25×6-ossal való összeszorzását, annak ellenére, hogy ennek a feladatnak a megoldásához 1200-nál kevesebb (625+150+150 = 925) tömbelem helyre lenne szükség.

A megoldás ötletét az (5.31) program adja: ne deklaráljunk három egyenként 400 elemű tömböt, hanem egyetlen 1200 eleműt, és ezt az aktuális méretektől függően bontsuk fel három

* Ha az aktuális és formális tömb sorainak száma megegyezik (kétméretű tömbre gondolunk), akkor az elemek megfelelkezése nem borul fel.

részre. Jelöljük ezt a tömböt XYZ-vel! Az (5.48) program megfelelően módosított változata a következő:

```
      MASTER PLD
      DIMENSION XYZ(1200)
      READ (...) L, M, N
      LIMIT = L*M
      DO 10 I=1, LIMIT1
      ...
      XYZ(I)=...
10    CONTINUE
      LIMIT2 = M*N
      DO 20 I=1, LIMIT2                                  (5.51)
      ...
      IND = I+LIMIT1
      XYZ(IND)=...
20    CONTINUE
      I = LIMIT1+1
      K = LIMIT1+LIMIT2+1
      CALL MTXMLT(XYZ(1), XYZ(I), XYZ(K), L, M, N)
      ...
      STOP
      END
```

Ha pl., mint kiindulási problémánkban, L=25, M=25 és N=6, akkor a mátrixszorzás szubrutinjában az A mátrix az XYZ tömb első eleménél kezdődik, és mérete 25×25-ös; a B mátrix az XYZ tömb 626-odik eleménél kezdődik és mérete 25×6-os, és végül a C mátrix az XYZ tömb 776-odik eleménél kezdődik és mérete ugyancsak 26×6-os. XYZ utolsó 275 eleme határo-

zatlan és nincs kihasználva. Egy ilyen szerkezetű program teljesen rugalmasan használható tetszőleges méretű tömbök összeszorzására; az egyetlen megkötés, hogy a feladatban szereplő tömbelemek száma ne haladja meg az 1200-at.

5. Közös adatmezők. A COMMON utasítás

Az előző két pontban megismerkedtünk az aktuális és formális paraméterek fogalmával, mint a több szegmensből álló programokban a belső kapcsolat fenntartásának eszközével. Láttuk, hogy a behívott szegmensek az aktuális paraméterekből veszik a bemenő információt, és az eredményeket ugyancsak a formális—aktuális paramétercsere közbejöttével adják vissza a behívó szegmensnek.

Ebben a pontban látni fogjuk, hogy a paramétercsere nem kizárólagos módja a szegmensek közti információcserének. A FORTRAN nyelv lehetőséget ad arra, hogy a program különböző szegmenseinek legyen egy vagy több közös „felvevő" és „lerakodó" helye — *közös adatmezeje* —, amelyhez a „társulásban résztvevő" szegmensek mindegyike hozzáférhet, és az akár saját maga, akár más szegmensek által oda elhelyezett értékeket a további számításokban felhasználhatja.

A közös adatmezők a FORTRAN nyelvben a COMMON utasítással deklarálhatók. Az utasítás alakja:

COMMON/BLOKK1/*1. lista*/BLOKK2/*2. lista*.../BLOKK*n*/*n. lista*

(5.52)

ahol BLOKK1, BLOKK2, ..., BLOKK*n* tetszőleges azonosítók („blokknevek"), az *1. lista*, *2. lista*, ..., *n. lista* pedig a típusdeklaráló utasításokban (lásd Negyedik fejezet 3. pontját) szereplő listákkal azonos konstrukciójú sorozatok. (Vagyis azonosítókból és tömbdeklarátorokból álló sorozatok.)

(5.52)-ben a BLOKK1, ..., BLOKK*n* azonosítók (blokknevek) bármelyike lehet üres, amely esetben az elválasztó törtvonalak között nem áll karaktersorozat. Kizárólag BLOKK1 esetében a két elválasztó törtvonal is elmaradhat, ha az azo-

nosító üres. Példák COMMON utasításokra:

```
| |COMMON/ADATOK/X(100), Y(20,20), Z/SZAMOK/
|X|PI, E, RYNOLD                                  (5.53)

| |COMMON/FILE/A(2500)//B(2500)                   (5.54)

| |COMMON//WORK(40,40)                            (5.55)

| |COMMON WORK(40,40)                             (5.56)
```

(5.53)-ban az első blokknév ADATOK, a második SZAMOK. (5.54)-ben az első blokknév FILE, a második üres, vagy más szóval *blank*. (5.55) és (5.56) teljesen egyenértékű, csupán az elsőben kitettük a két fölösleges törtvonalat, a másodikban nem. (Figyelem, a *blank* blokknév törtvonalai csak akkor hagyhatók el, ha az a COMMON utasítás első blokkneve. (5.54)-ben a két törtvonal nem fölösleges!)

A közös adatmezők deklarálásának lényegéhez tartozik, hogy ugyanaz a blokknév COMMON utasításokban a programnak egynél több szegmensében is előfordul. Valamely közös adatmező használatára azok a szegmensek „társulnak", amelyekben ugyanaz a blokknév COMMON utasításban előfordul. A közös adatmezők számára a fordítóprogram a memóriában egy-egy meghatározott részt jelöl ki, és ennek a résznek a rekeszeihez a „társulás"-ban résztvevő valamennyi szegmens hozzáférhet.

A blokknevet követő deklarációs lista a közös adatmezőhöz tartozó skaláris vagy indexes változókat határozza meg. A lista elemei által elfoglalt memóriarekeszek számát a közös adatmező *hosszának* nevezzük. Az (5.53) példában az ADATOK nevű adatmező hossza 501 egység, a SZAMOK nevű adatmező hossza 3 egység; (5.55)-ben, ill. (5.56)-ban a blank adatmező hossza 1600 egység. A közös adatmezők deklarálásának alapszabálya, hogy a nem blank adatmezők hosszát minden rájuk hivatkozó COMMON utasításban egyenlően kell megadni. (Blank adatmezőkre ez a megkötés nem vonatkozik.)

Példa

Tekintsük ismét az (5.35) és (5.38) programokban már vizsgált sh x-et generáló FUNCTION szegmenst! A közös adat-

mezőhöz tartozó (a továbbiakban röviden *common*) változók egy tipikus alkalmazási területe a mellékparamétereknek (mint pl. EPS és K) átadása, ill. átvétele. Az (5.35) program egy ilyen módosított változata lehet a következő:

```
FUNCTION SH(X)
COMMON/PARAMS/EPS, K
...                                  (5.57)
END
```

Természetesen a behívó szegmensnek is tartalmaznia kell a PARAMS nevű közös adatmező deklarációját:

```
MASTER BEHIVO
COMMON/PARAMS/EPS, K
...
EPS = 1.0E-06
G=SH(2.49666532)                     (5.58)
L = K/2+1
...
END
```

Az (5.58) BEHIVO szegmens (5.57) egy lehetséges aktivizálását mutatja. Mielőtt az SH szegmenst hívnánk, értéket adunk az EPS common változónak. Ezt az értéket az SH szegmens a számításban fel tudja használni. Ugyanakkor a szegmens végrehajtása közben keletkezett K értéket a BEHIVO program találja további felhasználásra készen a közös adatmezőben. Az utolsó utasítás a felhasználás módját illusztrálja: az L változó a sorfejtés tagjainak számát veszi fel értékként.

Bár az előbbi példa alapján erre lehetne következtetni, egyáltalán nem kötelező a közös adatmezőhöz tartozó változókat minden szegmensben ugyanazzal az azonosítóval ellátni. (5.58)-ban például ugyanolyan helyes volna — természetesen a szegmens többi utasításának átjelölése mellett — a következő COMMON utasítás is:

```
COMMON/PARAMS/ETA, N                 (5.59)
```

amely esetben az SH szegmens EPS változójának a BEHIVO szegmens ETA változója, SH K változójának pedig BEHIVO N változója felelne meg.

Ehhez hasonló egymáshoz rendeléssel már korábban is találkoztunk az EQUIVALENCE utasítással kapcsolatban (lásd a Negyedik fejezet 4. pontját). A lényeges különbség az, hogy míg az EQUIVALENCE mindig *ugyanazon* szegmens különböző változóinak azonosságát mondja ki, addig a COMMON utasítás *különböző* szegmensekben deklarált változók azonosságát deklarálja. Ezek a változók a különböző szegmensekben különböző nevekkel is lehetnek ellátva, de azonosságuk fokozott hangsúlyozása céljából sok esetben célszerű ugyanannak az azonosítónak a használata.

A COMMON utasítás felhasználható a közös adatmezőbeli tömbök indexhatárainak deklarálására is, az utasításban alkalmazott tömbdeklarátorok segítségével. Azonban a tömbméretek DIMENSION vagy típusdeklaráló utasításokban is deklarálhatók, akárcsak a nem közös adatmezőhöz tartozó — a továbbiakban *lokális* — változóké. Ha egy közös adatmezőhöz tartozó tömbnek a méreteit nem magában a COMMON utasításban deklaráljuk, akkor a közös adatmezőhöz tartozásának kinyilvánítására elegendő felsorolni a tömb azonosítóját a megfelelő blokknév után. Ezzel kapcsolatban emlékeztetünk arra a fontos szabályra (lásd a Negyedik fejezet 3c pontját), amely szerint minden tömb azonosítója egy szegmensben csak egyszer fordulhat elő tömbdeklarátor alkotórészeként.

Valamely közös adatmező *szerkezetén (struktúráján)* az illető adatmező felosztását értjük különböző skaláris, ill. indexes változókra. Például (5.53)-ban az ADATOK nevű közös adatmező egy 100 elemű vektorra, egy 20×20-as mátrixra és egy skaláris változóra van felosztva. A közös adatmezők alkalmazásának fontos szabálya, hogy *ugyanannak a közös adatmezőnek különböző szegmensekben eltérő szerkezete lehet.* Ez azt jelenti, hogy a programozónak jogában áll különböző szegmensekben ugyanazt a közös adatmezőt eltérő módon felosztani skaláris változókra és tömbökre. *Ilyen esetben a különböző szegmensek common változóinak egymáshoz rendelését a változóknak a közös adatmező kezdetéhez viszonyított relatív helyzete határozza meg.*

Példa

Tegyük fel, hogy valamely program A szegmensében az alábbi COMMON utasítás van elhelyezve:

```
| |COMMON/BLOKK/R(15,15), S(20,20), T(5,5)          (5.60)
```

A közös adatmező hossza $15\times15+20\times20+5\times5 = 650$ egység. Ezt a 650 egységnyi közös adatmezőt a B szegmensben a következő struktúrával is deklarálhatjuk:

```
| |COMMON/BLOKK/CSONT(10,10), BOR(20,20)
                                                  (5.61)
|X|SOR(150)
```

Az egyes indexes változók között a közös adatmezőben elfoglalt relatív helyzetük szerint a következő azonossági kapcsolatok jönnek létre:

1. Az A szegmens R(1,1) tömbelemének a B szegmens CSONT(1,1) tömbeleme felel meg;

2. A CSONT tömb (I, J)-edik elemének megfelel az R tömb valamely (I1, J1)-edik eleme, mindaddig, amíg „futja" a rövidebb CSONT tömb elemeiből;

3. Az R tömb 101-edik sorszámú R(11,7) elemének (lásd a sorszám-függvényt a Negyedik fejezet 3*d* pontjában) megfelel a BOR tömb (1,1) indexű eleme;

4. Az R tömb utolsó 125 elemének megfelel a BOR tömb első 125 eleme;

5. Az S tömb első 275 elemének megfelel a BOR tömb utolsó 275 eleme;

6. Az S tömb hátralevő 125 elemének megfelel SOR első 125 eleme;

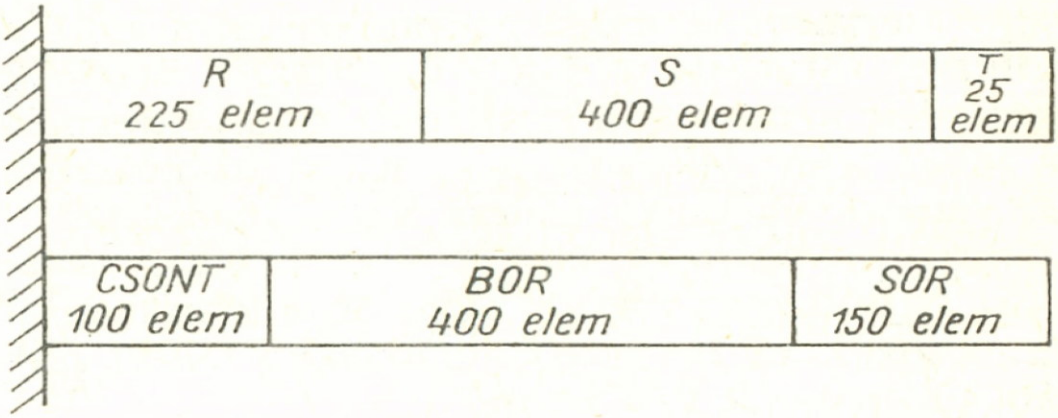
7. A T mátrixnak megfelel SOR utolsó 25 eleme.

A közös adatmező elrendezését az A és B szegmensben a 8. ábrán mutatjuk be.

Természetes, hogy ha különböző szegmensekben ugyanazt a közös adatmezőt különböző szerkezettel deklaráljuk, akkor az információ átadás—átvétel programozása során gondosan ügyelni kell a blokk változóinak azonossági kapcsolataira, nehogy programozási hiba következtében a másik szegmens bizonyos változóit elrontsuk.

Eddigi példáinkban hallgatólag feltettük, hogy az egymáshoz rendelt common változók mindegyik szegmensben azonos típusúak. Ha ez nem teljesül, akkor két dologra kell ügyelnünk:

1. Különböző típusú változók memória helyszükséglete általában *eltér* egymástól, ezért pl. ha egy COMPLEX típusú tömbnek egy másik szegmensben egy valós típusút feleltetünk meg, akkor *egy* komplex tömbelemhez a másik szegmensben *két* valós tömbelem tartozik.



8. ábra

Ha ilyen programozási fogásokkal kívánunk élni, feltétlenül tanulmányozzuk át annak a gépi reprezentációnak a kézikönyveit, amelyben a programot írjuk, mert a gépi reprezentációs nyelvek ebből a szempontból nem egyöntetűen viselkednek.

2. Különböző típusú változók egymáshoz rendelése általában azzal a következménnyel jár, hogy az „átvevő" szegmens — más ábrázolási módja miatt — nem tudja minden további nélkül felhasználni az „átadott" információt. Ha pl. az egyik szegmens INTEGER típusú változójának a másik szegmensben valós változó felel meg, akkor a közös adatmezőben átadott egész számot a másik szegmens általában nem tudja valósként használni.

Néhány további szabály a közös adatmezőkkel kapcsolatban:

1. A közös adatmezők nevei az egyéb azonosítókkal szemben kivételezett helyzetben vannak, amennyiben ők megegyezhetnek a szegmens bármelyik változójának, tömbjének vagy utasítás függvényének azonosítójával. Teljesen korrekt pl. a

következő deklaráció:

```
INTEGER BLOKK1
DIMENSION ADAT(50)                    (5.62)
COMMON/BLOKK1/ADAT
```

A BLOKK1 egész változónak semmi köze sincs a BLOKK1 nevű közös adatmezőhöz. A fordítóprogram előfordulásuk szerint meg tudja különböztetni, hogy mikor melyikről van szó.*

2. Egy szegmens formális paramétere nem lehet egyszersmind common változója is. Ha a COMMON utasításban formális paraméter azonosítója fordul elő, a fordítóprogram szintaktikus hibát jelez. *Aktuális* paramétere természetesen lehet common változó valamely eljárás aktivizálásának.

3. Common változók között nem szabad ekvivalenciát deklarálni. A szegmens *lokális* változói lehetnek ekvivalensek common változókkal, de az erre vonatkozó külön szabályokat (lásd alább) figyelembe kell venni.

4. Common változó kezdőértékét DATA utasítással csak BLOCK DATA szegmensben lehet beállítani. (BLOCK DATA szegmensekre nézve lásd a 6. pontot.)

5. Ugyanaz a blokknév egy szegmensen belül több COMMON utasításban is előfordulhat. Ilyen esetben az utóbb előforduló deklarációt a fordítóprogram az előző folytatásának tekinti. Tehát a

```
COMMON/NEV/A(50,50)
                                      (5.63)
COMMON/NEV/B(25,25)
```

deklarációpár egyenértékű a következővel [vö. (4.3) és (4.4)]

```
COMMON/NEV/A(50,50), B(25,25)         (5.64)
```

6. Ugyanaz az azonosító, ha nem blokknév, nem fordulhat elő egy szegmensen belül egynél több COMMON utasításban.

* Az azonosítók felhasználására vonatkozó szabályokat a 8. pontban külön is összefoglaljuk.

7. A *nem blank* közös adatmezőket minden rájuk hivatkozó szegmensben azonos hosszúságúként kell deklarálni. A *blank* közös adatmezők hossza a különböző szegmensekben eltérő is lehet.

Egyes gépi reprezentációkban a nem blank közös adatmezők is deklarálhatók eltérő hosszal; ebben az esetben a fordítóprogram a maximális hosszt veszi figyelembe.

Ha az ekvivalenciacsoportban common változó is van, akkor az EQUIVALENCE utasítás hatása lényegesen eltér a Negyedik fejezet 4. pontjában leírt sémától. A szabályok a következők:

1. Minden báziscsoportban (vö. Negyedik fejezet 4. pont) *legfeljebb egy* common változó lehet.

2. Ha az ekvivalenciacsoportban van common változó, akkor mindig ez a bázis. Ebben az esetben nem okvetlenül teljesül az az egyébként előírt követelmény, hogy a csoport többi elemeinek kezdete a bázis kezdetétől „jobbra" esik.

3. Az ekvivalencia nem bontja meg a közös adatmező struktúráját (a szabály magyarázatát lásd alább).

4. Az ekvivalencia meghosszabbíthatja a közös adatmezőt. Ilyen esetben a fenti 7. szabály alkalmazásakor a közös adatmező deklarált hosszán az ily módon meghosszabbított hosszt értjük.

5. Tilos olyan ekvivalenciát deklarálni, amely a közös adatmező kezdetének eltolását igényelné.

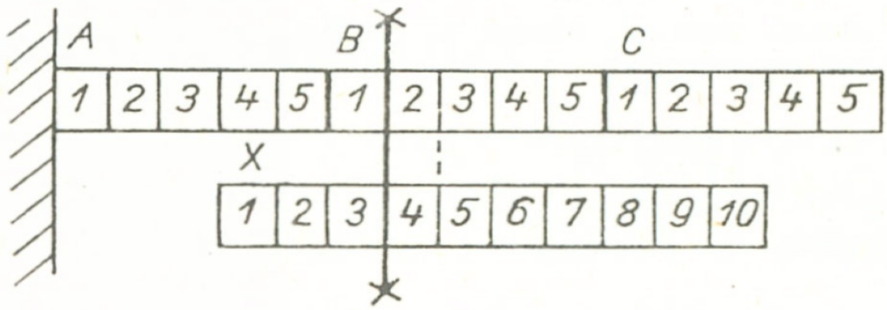
A következőkben példákkal illusztráljuk a különböző szabályokat.

A 3. szabály azt mondja ki, hogy — eltérően a lokális változók ekvivalenciájának esetétől — a közös adatmezőben fellépő ekvivalencia „nem tolja szét" a közös adatmező tömbjeit. Tekintsük pl. a következő deklarációkat:

```
DIMENSION X(10)
COMMON/PELDA/A(5), B(5), C(5)          (5.65)
EQUIVALENCE (X(4), B(2))
```

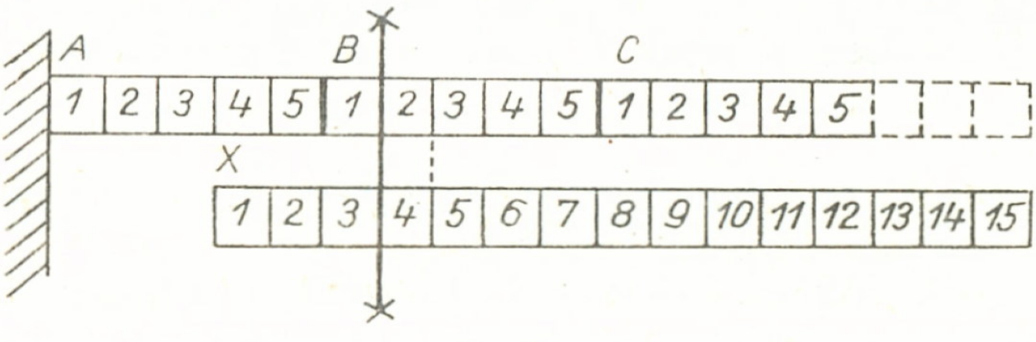
Ennek hatására nemcsak az X és B vektorok lépnek ekvivalenciába, hanem az X és A, ill. X és C vektorok is, hiszen a COMMON utasítás B-hez képest A és C helyzetét meghatározza

(9. ábra). Nevezetesen, az X vektor első eleme ekvivalens A(4)-gyel, X(2) ekvivalens A(5)-tel stb., továbbá X(8) ekvivalens C(1)-gyel, X(9) C(2)-vel stb. Az eltérés a lokális változók esetétől szembetűnő, (vö. 111. old. 3. szabály).



9. ábra

A 4. szabályra úgy kapunk példát, hogy (5.65)-ben X-et 10 helyett 15 eleműként deklaráljuk. Ebben az esetben az X vektor vége „kinyúlik" a PELDA közös adatmezőből. A közös adatmező meghosszabbítása azt jelenti, hogy a valódi méret nem 15, hanem 18 egység lesz. Szemléltetésül lásd a 10. ábrát!



10. ábra

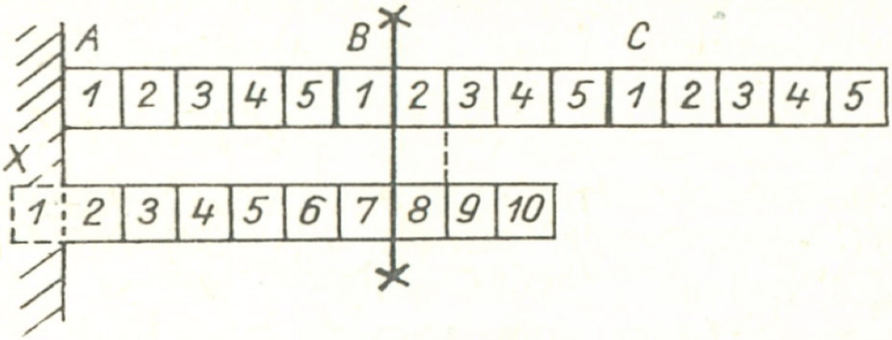
Végül az 5. szabályba ütköző — hibás — ekvivalenciához jutunk, ha (5.65)-ben az utolsó utasítás helyébe az

```
| |EQUIVALENCE(X(8), B(2))
```

utasítást írjuk. Ebben az esetben az X vektor eleje „kilóg" a közös adatmező kezdete elé. A közös adatmezőt bal felé nem lehet megnyújtani. Ezt az esetet a 11. ábra szemlélteti.

Láttuk az előzőkben, hogy a közös adatmezők használata alternatív lehetőség az aktuális—formális paramétercseréhez képest a szegmensközi információcsere megszervezésére. *Mindaz, ami közös adatmezőkkel megoldható, megoldható aktuá-*

lis—formális paramétercserével is. (Ezen állítás fordítottja nem érvényes!) Jogos tehát a kérdés, hogy milyen szempontok szerint választhatjuk egyik vagy másik megoldást, mikor melyik a célszerűbb?



11. ábra

Előrebocsátjuk, hogy erre merev szabályt adni nem lehet, és az alábbiakban valóban csak *célszerűségi* szempontokat adunk. Ha a programozó nem is tartja magát ezekhez, ettől programja még kifogástalanul működhet, legfeljebb kissé kevésbé lesz hatékony.

Az elsődleges választási szempont mindig a tárgyprogram térbeli (helyfoglalás szerinti) és időbeli (vagyis gépidő felhasználás szerinti) optimalizálása. Ezen belül általában a gépidő a fontosabb, hacsak a program helyfoglalása nem olyan nagy, hogy már fennáll a memóriakapacitás túllépésének veszélye. A közös adatmezők mind gépidő, mind helyfoglalás szempontjából sok esetben gazdaságosabbak a paramétercserénél. A fordítóprogram ugyanis a paraméterátvétel szervezésére utasításokat fordít a célprogramba, amelyek növelik annak hosszát, végrehajtásuk pedig időbe kerül. Ilyen utasítások beépítésére közös adatmezők esetén nem kerül sor, ezért a tárgyprogram gyorsabb és rövidebb lehet.

Egyúttal figyelembe kell azonban azt is venni, hogy a paramétercsere — rendeltetése folytán — módot ad az átjelölésre, tehát ugyanazt a számítási eljárást különböző azonosítóval ellátott objektumokon is el lehet végezni. (Lásd pl. az (5.25) és egyéb példákat.) Ha ugyanezt a feladatot közös adatmezőhöz tartozó változókkal vagy tömbökkel akarjuk megoldani, általában értékadó utasításokat kell végrehajtatnunk, amelyek a megfelelő változókat a közös adatmezőbe továbbítják. Ha tömbökről van szó, ez többszörös ciklusok beépítését teheti szükségessé, ami általában már nem gazdaságos. Ilyenkor tehát

kívánatosabb a paramétercserével dolgozni, mert a *tömbparaméterek átvétele nem jár az összes tömbelem áthelyezésével.*

További szempontot ad a dinamikus indexhatárok alkalmazásának lehetősége (lásd a 4. pontot). Tudjuk, hogy dinamikus indexhatárokat csak formális paraméter tömbnek lehet adni, common tömbnek nem. Amennyiben tehát dinamikus tömbméret megállapítása válik szükségessé, az ilyen tömböt okvetlenül formális paraméterként kell kezelni.

Összefoglalva, ha egy szegmens egy programban valamennyi aktivizálása során mindig ugyanazon változón vagy tömbön végez műveleteket és eközben mérete sem változik, akkor az ilyen változót vagy tömböt célszerűbb közös adatmezőbe tenni. Ha viszont a szegmens különböző aktivizálások során más és más azonosítóval ellátott objektumokon operál, vagy a tömbméretet dinamikusan kell deklarálnunk, akkor a tömböt formális paraméterként kezeljük.

Ehhez még annyit kell hozzátennünk, hogy „közhasználatra" bocsátott matematikai eljárások, szubrutinok FORTRAN programjaiban lehetőség szerint kerüljük közös adatmezők deklarálását, mert az ilyen szubrutin beépítésekor külön kell gondoskodnunk a behívó szegmensekben az azonos nevű közös adatmező deklarálásáról.

Néhánysoros illusztratív programokban a fenti elveknek természetesen szinte semmi jelentőségük nincs. Utalunk viszont a Hatodik fejezet *6b* pontban közölt teljes — és már nem egészen rövidlélegzetű — programra, amelynek analízise során kitérünk majd a „common vagy formális" probléma ott alkalmazott megoldására.

6. Közös adatmezőkben szereplő változók kezdőértékének beállítása. BLOCK DATA szegmensek

Említettük az előző pontban, hogy common változóknak közönséges programszegmensekben nem szabad DATA utasítással értéket adni. A jelen pontban a common változók kezdőértékének megadási módját ismerjük meg.

A common változók kezdőértékének beállítására a FORTRAN nyelv egy speciális szegmensfajtát definiál, az ún.

BLOCK DATA szegmenst, melynek az a specialitása, hogy nem tartalmaz végrehajtható utasításokat. A szegmensnyitó utasítás

BLOCK DATA (5.66)

Mint látható, ez az utasítás annyiban is különleges, hogy nem szerepel benne szegmensnév.

A fordítóprogramok — a gépi reprezentációtól függő módon — a BLOCK DATA szegmens részére fiktív szegmensnevet generálnak, amely pl. megegyezhet a szegmensben szereplő első közös adatmező nevével.

A szegmensnyitó utasítást követően a BLOCK DATA szegmens ugyanúgy folytatódik, mint egy közönséges szegmens: szabályos deklarációs rész következik, mely értelemszerűen mindig tartalmaz COMMON és DATA utasításokat. A deklarációs részt közvetlenül az END utasítás követi; végrehajtható utasítások a szegmensben nem lehetnek.

Példák

1. Tegyük fel, hogy az XXXX nevű közös adatmezőbe kívánunk kezdőértékeket elhelyezni: az X tízelemű vektor minden egyes elemét 1.0-val, az Y ötelemű vektor elemeit pedig rendre 1.0; 2.0; 3.0; 4.0 és 5.0 értékekkel kívánjuk feltölteni. A feladatot a következő BLOCK DATA szegmens oldhatja meg:

```
      BLOCK DATA
      COMMON/XXXX/X(10), Y(5)
      DATA X(1), X(2), X(3), X(4), X(5), X(6),
     1X(7), X(8), X(9), X(10)/10*1.0/,                  (5.67)
     2Y(1), Y(2), Y(3), Y(4), Y(5)/1.0, 2.0, 3.0,
     34.0, 5.0/
      END
```

Ez a példa némileg rámutat az ASA FORTRAN DATA utasításának hiányosságára: a beállítandó változók teljes listáját fel kell sorolni, nem írhatunk a listába tömbazonosítót. A gépi reprezentációk ezt általában

pótolják; többnyire megengedett a következő tömör írásmód is:

```
BLOCK DATA
COMMON / XXXX / X(10), Y(5)
DATA X/10*1.0/Y/1.0, 2.0, 3.0, 4.0, 5.0/
END
```

(5.68)

Sőt, van olyan gépi reprezentáció is, amelyben a DATA utasításban megengedettek ún. összetett listaelemek, amelyeket az ASA FORTRAN ki- és beviteli utasításokban használ (lásd a Hatodik fejezet 2. pontját).

2. Szerepeljen a program valamelyik — nem BLOCK DATA — szegmensében a következő deklaráció:

```
SUBROUTINE MASIK (...)
...
COMMON/BLOKK/A(5), B(5), C(5)
...
```

(5.69)

Kezdőértéket akarunk adni az A vektor első és a C vektor utolsó elemének: A(1)-nek az 5.0, C(5)-nek pedig a 3.35 értéket. A feladat BLOCK DATA szegmens írása, amely elvégzi ezt az értékadást.

Megoldás:

```
BLOCK DATA
DIMENSION B(13)
COMMON/BLOKK/A, B, C
DATA A, C/5.0, 3.35/
END
```

(5.70)

Az (5.70) példa több dolgot kíván illusztrálni. Először is emlékeztet arra, hogy különböző szegmensekben — és ez a BLOCK DATA szegmensre is vonatkozik — az azonosítókat eltérő módon is használhatjuk, és jogunk van a közös adatmezők „átszervezéséhez" is. Mint látható, éltünk is ezzel a joggal, és az A és C azonosítókat vektor helyett skalárok azonosítására használtuk fel, a B vektort pedig „átméreteztük". A BLOCK DATA szegmensbeli A, a közös adatmezők meg-

felelkezési szabályai szerint, a MASIK szegmens A(1)-ének, a BLOCK DATA-beli C pedig MASIK C(5)-jének felel meg (vö. (5.60)—(5.61)). A DATA utasítás tehát a megfelelő helyre teszi be az értékeket.

A másik érdekesség, amelyre fel kívánjuk hívni a figyelmet, a B azonosítónak puszta jelenléte a BLOCK DATA szegmensben. Észrevehetjük, hogy B nem lép fel a DATA utasításban, egyetlen feladata a helykitöltés. Alkalmazása nem meglepő, hiszen eleve tudjuk (165. old., 7. szabály), hogy a nem blank közös adatmezőket mindig azonos méretűnek kell deklarálni, és B ezt biztosítja. Látjuk tehát, hogy ha egy közös adatmező nem minden változójának akarunk kezdőértéket adni, a BLOCK DATA szegmensben helykitöltő változók deklarálása válhat szükségessé. Egyébként nemcsak BLOCK DATA szegmenseknek lehetnek helykitöltő változói; előfordulhat, hogy valamely szegmens egy közös adatmezőt csak részben használ, mégis — a méretek egyeztetése és a változók közti helyes megfelelkezés biztosítása érdekében — helykitöltő változókat kell deklarálnunk.

A BLOCK DATA szegmensben ekvivalenciákat is deklarálhatunk, de erre ritkábban van szükség. Ekvivalenciák általában úgy kerülnek a BLOCK DATA szegmensbe, hogy valamelyik másik szegmens deklarációs részét egyszerűen bemásoljuk a BLOCK DATA utasítás alá, és ezt egészítjük ki a kezdőérték adó DATA utasítással, (Ez ugyanis a BLOCK DATA szegmens készítésének egyik standard technikája. Természetesen olyan szegmenst kell választani, amelyben minden érintett közös adatmező szerepel.)

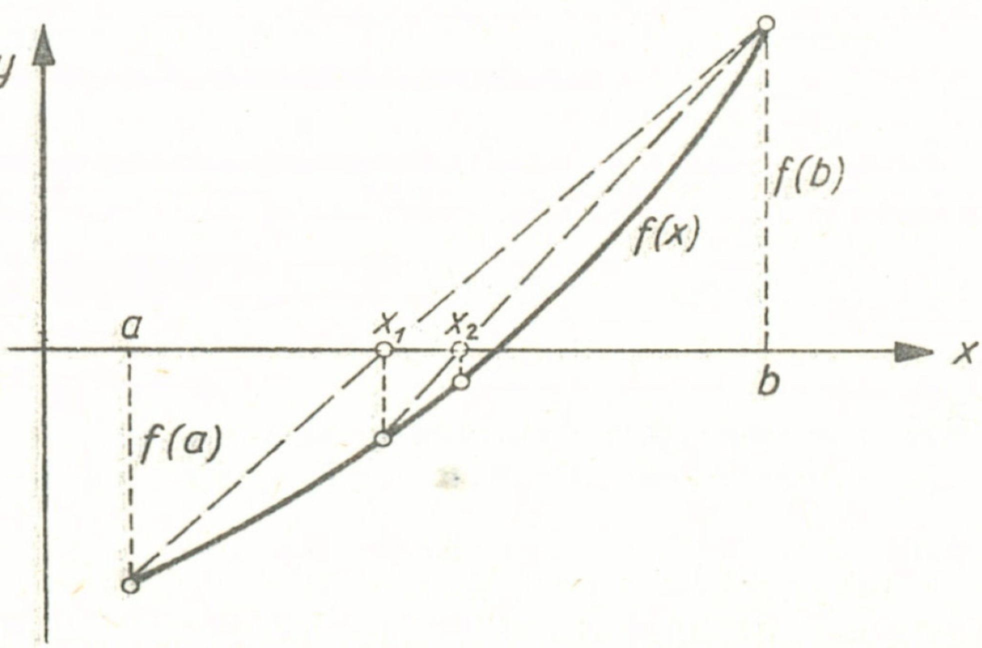
Kezdőérték adást az ASA FORTRAN csakis *nem blank* adatmezőkhöz tartozó változókra nézve enged meg. *Blank közös adatmezőkhöz tartozó változóknak kezdőértéket adni még* BLOCK DATA *szegmensben is tilos.*

7. Szegmensnevek alkalmazása aktuális és formális paraméterként. Az EXTERNAL utasítás

Eddigi tárgyalásainkban csupa olyan SUBROUTINE és FUNCTION szegmenst láttunk, amelynek formális paraméterei skaláris változók vagy tömbök, bár a 3. pontban jeleztük,

hogy a formális, ill. aktuális paraméterek között *szegmensazonosítók* is előfordulhatnak. Ebben a pontban szegmensazonosítók paraméterként történő felhasználásának szabályaival ismerkedünk meg.

Felvetődik a kérdés, hogy mi teszi egyáltalában szükségessé szegmenseknek paraméterként történő alkalmazását. Ennek szükségessége leggyakrabban úgy merül fel, hogy egy olyan szubrutint kívánunk írni, amely valamely előre nem meghatározott függvényen operál. Ilyen például egy transzcendens egyenletet megoldó program, amely tetszőleges megoldandó egyenlet esetén működésképes kell hogy legyen. Egy ilyen szubrutin természetesen hívja a bal oldali függvényt generáló FUNCTION szegmenst. Szerepelnek tehát benne F(X) alakú függvényhivatkozások, azonban az, hogy milyen konkrét függvényt jelent az F azonosító, csak akkor határozódik meg, amikor mellékeljük az F-et generáló szegmenst. Első pillanatban úgy tűnik, hogy ezt a feladatot eddigi ismereteink alapján is meg tudjuk oldani, hiszen nem kell mást tennünk, mint megírni az F FUNCTION szegmenst, és hozzácsatolni az egyenletmegoldó szubrutinhoz. Ha azonban ugyanazon programon belül ugyanazt az egyenletmegoldó szegmenst két különböző bal oldali függvényre — mondjuk F-re és G-re — alkalmazni kívánjuk, már baj van, mert ha az egyenletmegoldó szegmensben a bal oldali függvényt F-fel jelöltük, nem tudjuk megértetni a géppel, hogy másodszor ne az F-en, hanem a G-n operáljon.



12. ábra

Ezzel tehát ismét elérkeztünk a korábbról már ismerős „átkeresztelési" problémához, amelynek megoldását, mint láttuk, az aktuális—formális paramétercsere adta meg.

Lássunk mindjárt egy konkrét példát! Írjunk SUBROUTINE szegmenst, amely a húrmódszer alapján megoldja az $f(x)=0$ egyenletet. Mint ismeretes, a húrmódszer alkalmazásához két kezdőértékre van szükség, mondjuk a-ra és b-re, mégpedig úgy, hogy $f(a)>0$ és $f(b)<0$ (vagy fordítva) teljesüljön. A húrmódszer iterációs eljárás, amelyben a gyök soron következő közelítését a függvény e két pontját összekötő húr és az x-tengely metszéspontjaként állítjuk elő (lásd a 12. ábrát!). Az iterációs formula a következő:

$$x_{i+1} = \frac{x_i f(b) - b f(x_i)}{f(b) - f(x_i)} \qquad (i=1, 2, \ldots; \quad x_0 = a) \qquad (5.71)$$

Az (5.71) formulához fel kell tenni, hogy $f(x)$ első és második deriváltja az $[a, b]$ intervallumban sehol sem tűnik el, és hogy $f(a)$ és $f(x_1)$ egyező előjelűek (ellenkező esetben (5.71)-ben b helyett a írandó). Az (5.71) iteráció a „lassan, de biztosan" konvergáló eljárások körébe tartozik. Az ezt az eljárást realizáló FORTRAN szegmens* a következő lehet:

```
      SUBROUTINE SOLVE(A, B, X, F, EPS, N, SIKER)
      LOGICAL SIKER
      SIKER =.TRUE.
      N =0
      FA =F(A)
      FB =F(B)
      IF(FA*FB) 1, 2, 3
    3 SIKER =.FALSE.
```

* E szegmens listájában a működés jobb megértése céljából comment sort (lásd Első fejezet 3. pontját!) is alkalmaztunk. Ezt jelzi a sor bal szélső pozícióján álló C betű.

```
      RETURN
    2 X=A
      IF(ABS(FB).LT.ABS(FA))X=B
      RETURN
    1 N=1                                              (5.72)
      XE = (A*FB-B*FA)/(FB-FA)
      BX=B
      IF(F(XE)*FA) 4, 5, 5
    4 BX=A
      FB=FA
C     ELOKESZITES VEGE, ITERACIO KEZDETE
    5 N = N+1
      FE=F(XE)
      X = (XE*FB-BX*FE)/(FB-FE)
      IF(ABS((XE-X)/X).LT.EPS) RETURN
      XE=X
      GO TO 5
      END
```

Az (5.71) iterációs formula egyszerűsége ellenére a program nem olyan egyszerű, mint első pillanatban várná az ember. Ennek oka, hogy az eljárás megbízható működése érdekében a programba néhány aprólékos adminisztratív manipulációt is be kellett iktatnunk. Ezek egyike a kezdőértékek ellenőrzése. Ha az A és B kezdőértékek helyén a függvényértékek azonos előjelűek, az eljárás eredménytelenül azonnal véget ér. Az eredménytelenséget a SIKER logikai kimenő paraméter FALSE állapota jelzi. Ebben az esetben az X és N kimenő paraméterek értéke (utóbbi az iterációs lépéseket számlálja) határozatlan marad.

Amennyiben véletlenül az A és B kezdőértékek valamelyike éppen a pontos gyök volna, a program nem kezd iterálni, hanem szintén azonnal kilép a szegmensből, de ezúttal természetesen SIKER egyenlő TRUE-val, hiszen a gyökmeghatározás eredményes volt. Minthogy azonban iterációra nem került sor, az N kimenő paraméter értéke zérus.

Az iteráció az 5-ös címkéjű programrésszel kezdődik, és addig tart, ameddig az

$$\frac{x_{i+1}-x_i}{x_{i+1}} < \varepsilon$$

egyenlőtlenség be nem következik. Az X változó a mindenkori x_{i+1}-et, az XE változó pedig a mindenkori x_i-t tárolja. A még korábbi iteráltak az iteráció folyamán nem őrződnek meg. Ennyit magáról az eljárásról, és most lássuk, hogyan lehet SUBROUTINE szegmenst a gyakorlatban használni?

Példa

Oldjuk meg a SOLVE egyenletmegoldó program segítségével a következő két transzcendens egyenletet:

$$xe^x - 1 = 0 \tag{5.73}$$

és

$$\operatorname{tg} x - x = 0 \tag{5.74}$$

(5.73)-nak $x=0.5$ és $x=1.0$ között, (5.74)-nek pedig $x=3.2$ és $x=4.6$ között van egy-egy gyöke, tehát kezdőértéknek ezek az értékek választhatók. A megkívánt pontosság mindkét gyökre legyen 10^{-6}, tehát ezt az értéket válasszuk EPS-nek. A megoldásokat a GYOK1, ill. GYOK2 változóknak, az iterációs lépésszámokat az N1, ill. N2 változóknak adjuk át. A „válaszparaméterek" legyenek a logikai típusú BOOLE1 és BOOLE2 változók. A megoldandó egyenletek bal oldali függvényeit generáló FUNCTION szegmensek azonosítói legyenek EQ1 és EQ2.

A feladat megoldásának programja:

```
FUNCTION EQ1(X)
EQ1 = X*EXP(X)-1
RETURN
END
```

```
      FUNCTION EQ2(X)
      EQ2 = SIN (X)/COS (X) - X
      RETURN
      END
      MASTER KOCSOG
      LOGICAL BOOLE1, BOOLE2                              (5.75)
      EXTERNAL EQ1, EQ2
      CALL SOLVE(0.5, 1.0, GYOK1, EQ1, 0.1E-5, N1,
     X BOOLE1)
      CALL SOLVE(3.2, 4.6, GYOK2, EQ2, 0.1E-5, N2,
     X BOOLE2)
      IF(.NOT.BOOLE1.OR.NOT.BOOLE2) STOP 01
      WRITE (...) N1, GYOK1, N2, GYOK2
      STOP
      ...
      END
```

A MASTER szegmens lényeges része a két CALL utasítás, amely kétszer egymásután behívja a SOLVE húrmódszerrel dolgozó szubrutint, először az (5.73), másodszor pedig az (5.74) egyenlet megoldására. Az F formális paraméternek első esetben EQ1, második esetben EQ2 felel meg. A program a kiszámított GYOK1 és GYOK2 megoldások, valamint az N1 és N2 lépésszámok kinyomtatásával ér véget.

Az (5.75) programban úgyszólván ránézésre minden világos, csupán a MASTER szegmens harmadik sorában találunk egy ismeretlen utasítást, az EXTERNAL-t. Kérdés, mit jelent ez, és hogyan kell használni?

Formálisan az EXTERNAL utasítás az EXTERNAL alap-

szóból és egy listából áll. A lista elemei szegmensazonosítók (nem okvetlenül FUNCTION szegmensek azonosítói).

Az EXTERNAL *listára azokat a szegmensazonosítókat kell felvenni, amelyek valamely szegmens aktivizálásában aktuális paraméterként szerepelnek.* EQ1 és EQ2 példánkban ilyen szegmensek. Jól jegyezzük meg, hogy az EXTERNAL utasítás mindig a *behívó*, nem pedig a *behívott* szegmensben szerepel.

Ez utóbbi szabály főként az ALGOL nyelv ismerői számára tűnhet szokatlannak, az ALGOL-ban ugyanis az EXTERNAL utasítás analogonja a specifikációs részben szereplő **procedure** specifikáció, ez azonban az aktivizált, és nem az aktivizáló eljárásban van elhelyezve.

Az EXTERNAL utasításra a CALL utasítások helyes lefordításához van szükség. Az aktuális paraméterként fellépő szegmensazonosítóról ugyanis a fordítóprogram semmi módon nem tudná felismerni, hogy az nem egy eddig nem szerepelt skaláris változó. A zavart az okozza, hogy amikor a szegmensazonosító aktuális paraméterként szerepel, elveszti azokat az ismérveket, amelyek alapján a fordítóprogram őt egyébként fel szokta ismerni. Az idegen szegmens azonosítójának ismérvei ui. a CALL alapszó (SUBROUTINE szegmensek esetén), ill. az azonosítót követő bal zárójel (FUNCTION szegmensek esetén). A bal zárójel természetesen utalhatna indexes változóra is, ebben az esetben azonban a változónak valahol kellene hogy legyen egy tömbdeklarációja. Ennek hiányában a fordítóprogram szegmensazonosítót vélelmez. (Ebből következik, hogy a tévedésből nem deklarált tömböket a fordítóprogram gyakran FUNCTION szegmens azonosítójának értelmezi, és nem jelzi hibaként.)

Amikor a szegmensazonosító aktuális paraméterként lép fel, az említett ismérvek egyike sem fordul elő, így szükség van valamiféle kiegészítő információra, amely közli a fordítóprogrammal, hogy itt szegmensazonosítóról van szó.

Természetesen, ha a fordítóprogramnak *egyszerre* adnák oda az összes lefordítandó szegmenst, akkor a szegmensek összessége alapján felismerné a szegmensazonosítókat. Csakhogy éppen külön-külön adják a szegmenseket fordítani, és a szegmensenkénti fordíthatóság elve következtében gondoskodni kell arról, hogy egyetlen magában álló szegmensről is helyes fordítás készüljön. Ez a pontos magyarázata az EXTERNAL utasítás bevezetésének.

Az EXTERNAL utasítás bárhol előfordulhat a szegmens deklarációs részében.

Mint mondottuk, az EXTERNAL listán csakis szegmensazonosítók szerepelhetnek. Emlékeztetünk rá, hogy az utasításfüggvények azonosítói *nem* szegmensazonosítók, minthogy az utasításfüggvények nem számítanak szegmensnek. Ezek azonosítója tehát nem szerepelhet EXTERNAL utasításban, és utasításfüggvény nem is adható meg aktuális paraméterként, ha a megfelelő formális paraméter FUNCTION szegmens azonosítója. Ezért, bármilyen rövidek is (5.75)-ben EQ1 és EQ2 programjai, nem deklarálhattuk azokat utasítás függvényként.

A standard könyvtári függvények közül a *külső standard függvények* azonosítói szegmensazonosítónak számítanak, és ezért *szerepelhetnek aktuális paraméterként* és EXTERNAL utasításban. Ezzel szemben a *belső standard függvények* azonosítói lényegében úgy viselkednek, mint az utasítás függvények, tehát nem lehetnek szubrutinhívás aktuális paraméterei, és *nem szerepelhetnek* EXTERNAL *utasításban*. Ez a külső és belső standard függvények között a programozót érintő egyetlen lényeges különbség.

8. Szabályok az azonosítók használatára és az aktuális értékek meghatározására

Az előző fejezetekben megismert szabályok nagyrészt tartalmazzák az azonosítók használatára vonatkozó rendelkezéseket, mégis hasznosnak véljük ezen szabályok külön összefoglalását. Ugyancsak hasznosnak látszik azoknak az effektusoknak az összefoglalása, amelyek meghatározhatnak — vagy éppenséggel határozatlanná tehetnek — egy azonosítóhoz rendelt értéket. Mivel a ki- és beviteli utasításoknak is lehet ilyen hatása, itt említünk meg néhány ezekre vonatkozó szabályt is, bár tárgyalásukra csak a következő — Hatodik — fejezetben kerül sor.

Mint tudjuk, az azonosítókat két alapcsoportba sorolhatjuk. Ezek:

a) Szegmensen belüli azonosítók;
b) Szegmensközi azonosítók.

A szegmensen belüli azonosítók jellemzője, hogy csak a

szegmens „határain" belül érvényesek, minden szegmens önálló értelmet tulajdoníthat nekik, és a különböző szegmensekben érvényes értelmezésük egymástól teljesen független.

A szegmensközi azonosítók ezzel szemben nem definiálhatók különböző szegmensekben egymástól függetlenül; egy adott szegmensben meghatározott definíciójuk más szegmensekre is kihat.

Az azonosítók használatára vonatkozó főbb szabályokat a 9. és 10. táblázatokban foglaltuk össze. A táblázatok oszlopaiban az utasítástípusok, soraiban pedig az azonosítófajták állnak. A sorok és oszlopok kereszteződésében álló „+" jel azt jelenti, hogy az adott azonosítófajta az adott utasítástípusban használható, a „—" jel pedig azt jelenti, hogy nem használható. A zárójelben álló számok a táblázathoz fűzött megjegyzések valamelyikére utalnak.

A változók értékének meghatározására vonatkozó szabályokat összefoglalandó, emlékeztetünk a változók értékét meghatározó operációkra. Ezek a következők:

a) értékadó utasítás végrehajtása;

b) DO utasítás végrehajtása és összetett cikluslista elem értelmezése (lásd Hatodik fejezet);

c) beolvasási utasítások végrehajtása (lásd Hatodik fejezet);

d) ASSIGN utasítás végrehajtása;

e) közös adatmezőn keresztül történő adatcsere;

f) ekvivalenciák által meghatározott megfelelkezés révén történő adatcsere;

g) aktuális—formális paramétercsere;

h) kezdőérték megállapítása DATA utasítás segítségével.

Az *a)—d)* alatti operációk közös jellemzője, hogy ezekben az érték meghatározása *explicit* módon megy végbe; az utasításban minden esetben magának a meghatározandó változónak az azonosítója szerepel. Ezzel szemben az *e)—g)* utasítások az érintett változó valamelyik hozzárendeltjének értékét változtatják meg és ez közvetve hat vissza az érintett változó értékére. Ez utóbbi esetben *implicit értékmeghatározásról* beszélhetünk. A *h)* alatti értékmeghatározás lehet explicit is és implicit is.

9. táblázat

A szegmensen belüli azonosítók használata

Azonosító használata / fajtája	Típus-deklaráló	DIMENSION	COMMON	EQUIVALENCE	DATA	EXTERNAL	Egyéb végrehajtható(7)	Aktuális paraméterként
	utasításban							
Formális paraméter								
Tömb és indexes változó	+ (1)	+ (1)	–	–	–	[illegible]	+	+
Skaláris változó	+	–	–	–	–	–	+	+
Szegmensazonosító	+ (2)	–	–	–	–	+	+ (4)	+ (3)
Nem formális paraméter COMMON								
Tömb és indexes változó	+	+	+	+ (5)	+ (6)	–	+	+
Skaláris változó	+	–	+	+ (5)	+ (6)	–	+	+
Nem formális paraméter és nem COMMON								
Tömb és indexes változó	+	+	–	+	+	–	+	+
Skaláris változó	+	–	–	+	+	–	+	+
Utasításfüggvény	+	–	–	–	–	–	+	–

Megjegyzések a 9. táblázathoz:

1. Dinamikus indexhatárok deklarálása is megengedett.

2. A formális paraméter szegmens azonosító szegmensen belülinek számít. A szegmens végrehajtásakor természetesen a megfelelő aktuális paraméter lép a helyébe, amely már szegmensközi azonosító.

3. EXTERNAL utasításban való deklarálása kötelező.

4. Be- és kiviteli utasítások kivételével.

5. A báziscsoportban csak egyetlen COMMON változó lehet.

6. Csak BLOCK DATA szegmensben, és csak akkor, ha a változó nem blank adatmezőhöz tartozik.

7. „Egyéb" végrehajtható utasításon itt értékadó, aritmetikai és logikai IF, be- és kiviteli, GO TO és DO utasítást értünk, és a „+" jel értelmezéséhez itt hozzáteendő: „a megfelelő utasítás szabályainak megfelelően".

10. táblázat

A szegmensközi azonosítók használata

Azonosító fajtája	SUBROUTINE FUNCTION	Típus deklaráló	EXTERNAL	Egyéb deklaratív	Egyéb végrehajtható[6]	Aktuális paraméterként
	utasításban					
Belső standard függvény	—	+[2]	—	+[1]	+[5]	—
Külső standard függvény	+[1]	+[2]	+	+[1]	+[5]	+[3]
FORTRAN szegmens azonosítója	+	+[2]	+	—	+	+
Közös adatmező neve	—	+[4]	—	+[4]	+[4]	+[4]

Megjegyzések a 10. táblázathoz:

1. Az azonosítótípus egyes szegmenseken belül külön is deklarálható, mint szegmensen belüli azonosító. Ilyen esetben használatára a 9. táblázat szerintiek az irányadók. Az azonosítónak az ilyen utasítástípusban való megjelenése annak szegmensen belüli — tehát rendeltetésétől eltérő — használatát jelenti.

2. Csak a függvény eredeti definíciójának ellent nem mondó típusdeklaráció megengedett, ellenkező esetben az azonosító újradeklarálásával állunk szemben (l. az 1. sz. megjegyzést).

3. EXTERNAL utasításban való deklarálása kötelező.

4. A közös adatmező neve egyszersmind szegmensen belüli azonosító is lehet, ennek azonban semmi köze sincs az azonosító szegmensközi jellegéhez. Blokknévként csak a COMMON utasítás blokknév meghatározó részében fordulhat elő, egyéb előfordulásaira nézve a 9. táblázatban foglaltak az irányadók.

5. Az azonosító eredeti definíciójának megfelelően.

6. Az „egyéb végrehajtható" utasításokat itt a be- és kiviteli utasítások kivételével értjük.

A 179. oldali felsorolásból az is látható, hogy változók egymáshoz rendelését az aktuális—formális paramétercsere, valamint a COMMON és az EQUIVALENCE utasítás hozhatja létre. A FORTRAN nyelv szabályai az explicit és az implicit értékmeghatározás hatását általában egyenértékűnek tekintik. Erre vonatkozik az az alapvető szabály, amely szerint *egy adott változó értékének explicit meghatározása maga után vonja valamennyi azonos típusú hozzárendelt változó értékének implicit meghatározását*. Hangsúlyozzuk, hogy ez csak az azonos típusú hozzárendelt változókra vonatkozik. Az eltérő típusú hozzárendelt változók értéke az explicit értékadással egyidejűleg meghatározatlanná válik.

Egyetlen megszorítás terheli az implicit értékadást: *indexként, valamint kijelölt* GO TO *utasításban használt változó értékét csak explicite szabad meghatározni;* ilyen célú felhasználás szempontjából az implicite meghatározott változó értéke határozatlannak tekintendő.

Az értékek meghatározására, ill. határozatlanná tételére vonatkozó fontosabb szabályok a következők:

1. Kifejezések kiértékelésekor a számításban érintett változók értékének meghatározottnak kell lennie.

2. DO ciklus végrehajtásának, vagy összetett listaelem (Hatodik fejezet) értelmezésének befejezése után a ciklusváltozó értéke határozatlan.

3. DO ciklus végrehajtása közben a ciklusparaméterek értékét megváltoztatni nem szabad.

4. CALL utasítás végrehajtásakor vagy FUNCTION szeg-

mensek aktivizálásakor az aktuális paraméterekben szereplő kifejezések értékének és az aktuális paraméterként megadott indexes változók indexének meghatározottnak kell lennie.

5. FUNCTION szegmens végrehajtásának befejezésekor (RETURN utasítás elérése) a függvényértéknek meghatározottnak kell lennie.

6. Kiviteli (WRITE) utasítások végrehajtásának kezdetén az összes átviendő változó értékének meghatározottnak kell lennie.

9. Gyakorlatok

1. Írjunk SUBROUTINE szegmenst négyzetes mátrix transzponálására, az $a_{ik}=\bar{a}_{ki}$ összefüggés alapján, ahol az indexelt változók mátrixelemeket, a felülhúzás pedig a transzponált mátrix elemeit jelöli. A szegmensben alkalmazzunk dinamikus indexhatárokat! Ezután írjuk meg a vázát egy olyan MASTER szegmensnek, amely előállítja egy 5×5-ös, egy 6×6-os és egy 7×7-es mátrix elemeit, majd rendre behívja a transzponáló szubrutint, és elvégezteti a transzponálást. (Útmutatás: tekintsük az (5.48) programot!)

2. Írjunk SUBROUTINE szegmenst egy N elemű vektor elemeinek nagyság szerint növekedő sorbarendezésére! A megoldásban felhasználhatjuk a (3.70) programrészletet. Írjunk ezután MASTER szegmenst, amely beolvastatja az N számot és a V vektor elemeit, sorba rendezteti azokat, majd az eredményvektor elemeit kiíratja (a be- és kiviteli utasításokat szimbolizáljuk a megszokott módon). A MASTER és a SUBROUTINE szegmens közötti adatcserét végeztethetjük közös adatmezőn keresztül, vagy aktuális—formális paramétercsere alapján. Az adott esetben melyik látszik célszerűbbnek? Miért? Mi a helyzet a dinamikus indexhatár használatával?

3. Írjunk SUBROUTINE szegmenst vektorok diadikus szorzatának előállítására! Mint ismeretes, az N elemű **u** és **v** vektorok diadikus szorzatán azt az $N \times N$-es mátrixot értjük, amelynek i, k indexű elemére

$$a_{ik} = [\mathbf{u}\mathbf{v}^*]_{ik} = u_i v_k \qquad i, k=1, \dots N \tag{5.76}$$

(Itt $\mathbf{v}^*$ a **v** oszlopvektor transzponáltját jelenti.)

4. Írjunk SUBROUTINE szegmenseket
a) a mátrixösszeadás,
b) a mátrixkivonás
műveleteire. A megfelelő képletek:

$$c_{ik} = a_{ik} \pm b_{ik} \qquad i, k=1, \dots, N \tag{5.77}$$

5. Alakítsuk át SUBROUTINE szegmenssé a Harmadik fejezet 10. feladatának (mátrix maximális abszolút értékű sajátértékének meghatározása) megoldását! A paraméterek legyenek A, X, Y, ALAM, ITMAX, EPS és N!

6. Kisebb rendszámú nemszinguláris szimmetrikus mátrix összes sajátértékét is meghatározhatjuk az előző feladat módszerével, a következő módon:

a) Kiindulunk az **A** mátrixból, és meghatározzuk ennek maximális abszolút értékű sajátértékét.

b) Jelöljük az előző lépésben kapott sajátértéket λ_1-gyel, a hozzá tartozó sajátvektort pedig $\mathbf{x}_1$-gyel! Képezzük az

$$\mathbf{A}^{(1)} = \mathbf{A} - \lambda_1 \mathbf{x}_1 \mathbf{x}_1^* \tag{5.78}$$

mátrixot! Bebizonyítható, hogy ennek sajátértékei λ_1 kivételével megegyeznek **A** sajátértékeivel; λ_1 helyett pedig zérus áll.

c) Határozzuk meg $\mathbf{A}^{(1)}$ maximális abszolút értékű sajátértékét és sajátvektorát az ismert módon, és jelöljük ezeket λ_2-vel és $\mathbf{x}_2$-vel! Alkalmazzuk az (5.78) formulát $\mathbf{A} = \mathbf{A}^{(1)}$-gyel, $\lambda_1 = \lambda_2$-vel és $\mathbf{x}_1 = \mathbf{x}_2$-vel!

d) Általánosan legyen

$$\mathbf{A}^{(i)} = \mathbf{A}^{(i-1)} - \lambda_i \mathbf{x}_i \mathbf{x}_i^* \tag{5.79}$$

Ismételjük az (5.79) iterációs formulát *N* lépésben, ekkor megkapjuk **A** összes sajátértékét, és a hozzá tartozó sajátvektorokat.

Írjunk általános sajátérték meghatározó SUBROUTINE szegmenst a fenti algoritmus alapján! Segédeljárásként használjuk fel a 3., 4. és 5. feladatok megoldásait! Kezdőérték-vektornak válasszuk mindenkor a mátrixnak azt a sorát, amelyben a maximális abszolút értékű diagonális elem áll. Az (5.79) iterációs eljárás lényegében az **A** mátrix ún. *diadikus felbontásának* egy numerikus módszere. Kimutatható, hogy *N* lépés után az egyenlet bal oldalán a mátrixkivonás eredménye a zérusmátrix, és ebből kifolyólag

$$\mathbf{A} = \sum_{i=1}^{N} \lambda_i \mathbf{x}_i \mathbf{x}_i^* \tag{5.80}$$

Az eljárás sajnos nem alkalmas széles körű felhasználásra, mert általában lassú és numerikusan instabil.

7. Magasabbfokú algebrai egyenletek megoldására — megfelelő kezdőértékek birtokában — az ún. Horner-módszer alkalmazható; ez azon alapszik, hogy egy $p(z) = a_1 z^n + a_2 z^{n-1} + \ldots + a_{n+1}$ polinom helyettesítési értéke valamely z helyen a következő rekurziós formulával határozható meg:

$$\begin{aligned} b_0 &= 0 \\ b_{i+1} &= a_{i+1} + z b_i \qquad (i = 0,\ 1, \ldots, n) \\ p(z) &= b_{n+1} \end{aligned} \tag{5.81}$$

Ez nem egyéb, mint a Horner-séma, amelyet a (3.61) programrészlet mutatott be. Könnyű belátni, hogy ha az (5.81) rekurziót az a_{i-k} helyett a b_i számokon is végrehajtjuk és ezzel egy újabb, c_i sorozatot generálunk:

$$\begin{aligned} c_0 &= 0 \\ c_{i+1} &= b_{i+1} + z c_i \qquad (i = 0, 1, \ldots, n-1) \\ q(z) &= c_n \end{aligned} \tag{5.82}$$

akkor a kapott $q(z)$ mennyiség éppen $p'(z)$-vel egyenlő. Ezt a tényt felhasználhatjuk a $p(z)=0$ egyenletnek Newton—Raphson-módszerrel történő megoldására. Mint ismeretes, a Newton—Raphson iterációs eljárás a $p(z)=0$ egyenlet gyökét a következő iterációval állítja elő:

$$z^{(k+1)} = z^{(k)} - \frac{p(z)^{(k)}}{p'(z)^{(k)}} \tag{5.83}$$

ahol $p(z)^{(k)}$ és $p'(z)^{(k)}$ előállítására az (5.81) és (5.82) rekurziós formulák alkalmazhatók. Ha ismeretes egy gyöknek valamely $z^{(0)}$ kezdeti közelítése, akkor ily módon előállíthatjuk a gyököt.

A módszer egyszersmind alkalmas az összes gyök előállítására is, ha ezek valósak és mind különbözők (valójában akkor is, ha nem minden gyök valós, de akkor az egész eljárást komplex aritmetikával kell végrehajtani). Az (5.81) rekurziós formulával kapott b_i számok ugyanis, ha $\bar{z}$ gyök, azaz $p(\bar{z})=0$, a $z-\bar{z}$ gyöktényezővel leosztott polinom együtthatóit adják meg, tehát a gyökkeresést a b_i együtthatójú polinomon folytatva, meghatározhatjuk az eredeti polinom egy újabb gyökét. Ismét csökkentve eggyel a fokszámot, végül eljuthatunk a polinom valamennyi gyökéhez. Írjunk SUBROUTINE szegmenst, amely végrehajtja a leírt eljárást! Formális paraméterek legyenek a következők: A legyen az együtthatókat tartalmazó vektor, N legyen a rendszám, XO legyen az egyes gyökök nulladik közelítését tartalmazó kezdőérték-vektor, X legyen az eredményvektor, és EPS legyen az előírt pontossági paraméter.

A leírt eljárás numerikus problémái hasonlók az előző feladatéhoz, ami nem meglepő, ha figyelembe vesszük a két feladat közti szoros analógiát. Többszörös gyök esetén az eljárás mindenképpen csődöt mond, mert akkor (5.83)-ban zérus kerül a nevezőbe.

8. Írjunk SUBROUTINE szegmenst az ún. szelőmódszerrel végzett gyökmeghatározáshoz! Mint ismeretes, a szelőmódszer az $f(x)=0$ transzcendens egyenlet megoldására szolgáló következő iterációs formulán alapul:

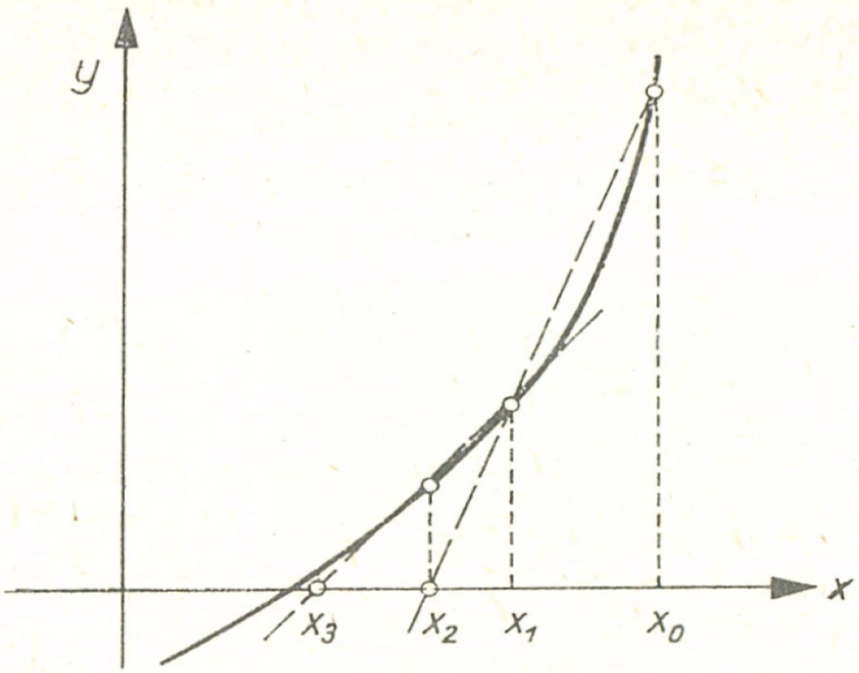
$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{y_i - y_{i-1}} y_i \tag{5.84}$$

ahol x_i a gyök i-edik iteráltja, és $y_i=f(x_i)$. Geometriailag ez azt jelenti, hogy a gyök újabb közelítése a két megelőző közelítés helyén vett függvényértékeken átfektetett szelő és az x-tengely metszéspontja. (Lásd a 13. ábrát!) A gyökmeghatározó szegmens formális paraméterei legyenek az XO és X1 két kezdeti közelítés, a megoldandó egyenlet bal oldalát generáló FUNCTION szegmens azonosítója, az előírt relatív pontosság EPS mérőszáma, a megengedett iterációs lépések maximális száma, és végül a ténylegesen elvégzett iterációs lépések száma.

Az eljárást úgy konstruáljuk meg, hogy ha az iteráció során a megengedett maximális számú lépésben nem érjük el a megkívánt pontosságot, akkor a gép álljon meg egy PAUSE 77 utasításon.

Ha ez a SUBROUTINE szegmens már megvan, írjunk programot az

$$f(x) = \sin x - \frac{x}{2} \tag{5.85}$$



13. ábra

transzcendens egyenlet megoldására! Ehhez szükségünk van egy FUNCTION és egy MASTER szegmensre. Írjuk meg ezeket a szegmenseket az (5.75) programok mintájára!

9. Írjunk SUBROUTINE szegmenst ún. trapéz-szabály alapján végzendő numerikus integrációra! A trapéz-szabály a következő:

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(a)+\sum_{i=1}^{n-1} 2f(a+ih)+f(b)\right] \tag{5.86}$$

ahol $h=(b-a)/n$, és n pozitív egész szám. Az n szám általában előre nem adott, mert annak à priori meghatározása a függvény menetének alapos ismeretét igényelné. A gyakorlatban tehát sokszor kell ún. „önosztó" eljárásokhoz folyamodnunk, amelyek automatikusan határozzák meg az integráció lépésközét.

Az ilyen önosztó eljárások alapgondolata az, hogy először kiszámítják valamilyen kezdeti lépésközzel az integrál egy közelítését, majd a lépésköz szukcesszív felezésével további, egyre pontosabb közelítéseket határoznak meg. Az, hogy a lépésközt felezzük, nem tartozik a dolog lényegéhez, alkalmazhatnánk egyéb lépésköz rövidítési algoritmust is, azonban — mint látni fogjuk — a felezés jelentős programozási kényelmet jelent.

Induljunk ki tehát valamely $n=2^{N_0}$ kezdeti osztópontszámból. Jelöljük a megfelelő integrálközelítést S_n-nel! Könnyű belátni, hogy ha a lépésközt megfelezzük (azaz az osztópontszámot megkétszerezzük), az integrál új közelítésének meghatározásához már csak a „közbülső" pontokban vett függvényértékeket kell kiszámítanunk, és hogy

$$S_{n+1}=\frac{S_n}{2}+\frac{h}{2}\sum_{i=1}^{n} f\left(a+\frac{h}{2}+(i-1)h\right) \tag{5.87}$$

ahol $h = (b-a)/n$. (5.87) tehát rekurziós formula az integrál egymás utáni közelítéseinek előállítására. Bebizonyítható, hogy az ily módon generált $\{S_n\}$ sorozat konvergál az integrál pontos értékéhez.

Írjunk SUBROUTINE szegmenst ennek a numerikus integrálási eljárásnak a realizálására! A szegmens bemenő paraméterei legyenek: az intervallum kezdő- és végpontja, a kezdeti lépésköz, az integrálandó függvény és az előírt relatív pontosság; az eredmény paraméterek pedig legyenek: az integrál értéke és az integrál előállításához szükséges osztópontok száma.

10. Az előző feladatban leírt numerikus integrálásnak továbbfejlesztett, és igen hatékony változata az ún. Romberg-féle eljárás, melynek lényege a következő: tekintsük a trapézszabály alapján nyert integrálközelítések sorozatát, és jelöljük ezeket $S_n^{(1)}$-gyel. Ezeket egy háromszögmátrix első oszlopának véve (a felső, zárójelbe zárt index jelenti az oszlopindexet), a háromszögmátrix további elemeit az alábbi rekurziós formula alapján számíthatjuk ki:

$$S_n^{(k+1)} = \frac{4^k S_n^{(k)} - S_{n-1}^{(k)}}{4^k - 1} \qquad n=1, 2, \dots; \qquad k=1, 2, \dots, n \tag{5.88}$$

(5.88)-ból látható, hogy ha már kiszámítottuk a mátrix első $n-1$ sorát, akkor ebből kiszámíthatjuk az n-ediket. (A képzési szabály némiképpen emlékeztet a Pascal-háromszög képzési szabályára.)

Az (5.88) rekurzióval képzett háromszögmátrixról bebizonyítható, hogy rögzített oszlopindex mellett $n \to \infty$ esetén $S_n^{(k)}$ tart az $f(x)$ függvény $[a, b]$ intervallumon vett határozott integráljához. Az is kimutatható, hogy az egy sorban álló elemek az integrál egyre magasabb rendű közelítései. A mátrix második oszlopának elemei a Simpson-féle integrációs formulával nyert közelítésekkel azonosak.

A leírt eljárást a következő módon használhatjuk fel valamely függvény határozott integráljának kiszámítására:

a) Előállítjuk az integrál trapézformulából számított közelítéseit, valamely előre rögzített NMAX paraméterértékig bezárólag.

b) Ha ezáltal elő tudtuk állítani az integrált az előírt pontossággal, akkor nincs több teendő. Ha nem, akkor az (5.88) rekurziós formula alapján előállítjuk rendre a háromszögmátrix sorait is. Ha valamelyik sor két egymás melletti eleme már elegendően kevéssé tér el, ugyancsak befejezettnek nyilvánítjuk az eljárást; ha ez még az NMAX indexű sornál sem következne be, akkor ez azt jelenti, hogy az adott osztópontszám mellett az integrált nem tudjuk az előírt pontosságra kiszámítani. Ebben az esetben vagy a pontossági igényt kell leszállítanunk, vagy az osztópontszámot növelnünk.

Amennyiben az S mátrix valamennyi elemét tárolni akarjuk, akkor kb. $NMAX^2/2$ számú memóriarekeszre van szükségünk. Valójában ennél kevesebb rekesszel is megoldhatjuk a feladatot, ha figyelembe vesszük, hogy a mátrix n-edik sorának előállításához csak az $n-1$-edik sorra van szükségünk, a többi sor elemei az n-edik sor elemeinek értékét csak implicite befolyásolják. Sőt, az $n-1$-edik sornak sincs minden elemére szükségünk, hanem csak azokra, amelyeknek oszlopindexe nagyobb vagy egyenlő, mint az éppen kiszámítás alatt álló elemé (lásd az (5.88) rekurziós

formulát; az egyes sorok elemeit balról jobbra tartó sorrendben, azaz növekvő oszlopindexek szerint számítjuk ki).

A gazdaságos számítási algoritmus tehát a következő. Valamely NMAX elemű T vektor elemeiben elhelyezzük az $S_i^{(1)}$ integrálközelítéseket:

$$T_i = S_i^{(1)} \qquad (i=1, \ldots, \mathrm{NMAX}) \tag{5.89}$$

Ezután alkalmazzuk az (5.88) rekurziós formulát $k=1$-gyel és $n=1$-gyel, és képezzük $S_2^{(2)}$-t. Mivel a továbbiakban $S_1^{(1)}$-re már nem lesz szükségünk, írjuk be $S_2^{(2)}$-t $S_1^{(1)}$ helyére, a T vektor első elemébe. Az $S_3^{(2)}$ mátrixelem kiszámításához $S_3^{(1)}$-re és $S_2^{(1)}$-re van szükségünk, azaz T_3-ra és T_2-re. A továbbiakban $S_2^{(1)}$-re már nem lesz szükségünk, így a helyére T_2-be írhatjuk $S_3^{(2)}$-t. $S_3^{(3)}$ előállításában $S_3^{(2)}$ és $S_2^{(2)}$ szerepel, azaz T_2 és T_1. Ezt a későbbiekben már szükségtelen $S_2^{(2)}$ helyére, T_1-be írhatjuk be s. í. t. Látható, hogy ez az eljárás az i-edik sor előállítása során a T vektor első i elemébe helyezi az $S_i^{(k-i+1)}$ elemeket ($k=1, \ldots, i$). Ha ezt végrehajtjuk $i=1, \ldots,$ NMAX-ra, az utolsó lépésben megkapjuk a Romberg-mátrix NMAX indexű sorát. Ha eközben a T vektor két szomszédos eleme az előírt ε relatív pontosságnál kevésbé tér el, a táblázat elemeinek kiszámítását előbb is abbahagyhatjuk.

Az algoritmusról megjegyezzük még, hogy az a T_{NMAX} elemet nem rontja el, így — amennyiben NMAX lépésben nem értük el a kívánt pontosságot — ezen utolsó elem felhasználásával számíthatunk egy újabb trapéz-integrált (lásd az előző, 9-es feladatot), és ezzel az újabb elemmel továbbfolytathatjuk a Romberg-mátrix generálását.

Írjunk SUBROUTINE szegmenst a Romberg-eljárásra! A bemenő paraméterek legyenek: az integráció határai, az integrálandó függvény, az ε kívánt relatív pontosság és végül a megengedett NMAX maximális lépésközszám. Kimenő paraméterek: az integrál értéke, az LA és az LB logikai változók, melyek közül LA TRUE értékű, ha az integrált NMAX lépésben ε pontossággal rendben kiszámítottuk, LA=FALSE és LB=TRUE, ha NMAX lépésben ugyan nem értünk célt, de NMAX növelésével még sikerült a kívánt pontosságot elérni, és végül LB=FALSE, ha N=1024 sem volt megfelelő, amin túl NMAX nem növelhető, mert a T vektort 10 eleműre deklaráljuk. ($2^{10}=1024$.) Ha a behíváskor mindjárt NMAX>1024, akkor az eljárás állítsa FALSE-ra LB értékét és azonnal lépjen ki. LB=FALSE esetén az integrál értékét tartalmazó kimenő paraméter értéke határozatlan marad.

Ha az integráló szubrutin már megvan, írjunk teljes programot az

$$I = \int_0^1 x^2 e^x \, dx; \qquad \varepsilon = 10^{-6} \tag{5.90}$$

határozott integrál numerikus kiszámítására! Legyen NMAX=64.

Hatodik fejezet

ADATÁTVITEL ÉS UTASÍTÁSAI

1. Perifériális egységek és külső tárolók

Könyvünk előző fejezeteiben megismertük a különböző FORTRAN utasításokat, valamint az adat- és változó deklarációk módozatait. Egy igen fontos témakör azonban még hátra van, mégpedig a perifériális egységekkel és a külső tárolókkal való adatcsere műveleteinek ismertetése. A megfelelő FORTRAN utasításokat a következő ponttól kezdve tárgyaljuk; a jelen pontban kissé elkanyarodunk a FORTRAN nyelvtől, és a momodern számológépek perifériális egységeinek néhány olyan tulajdonságát foglaljuk össze, amelyek ismeretére a későbbiekben szükségünk lesz.

Régebben általában élesen megkülönböztették, és a gép külön funkcionális egységének tekintették az ún. „perifériális" egységeket (lyukszalag- vagy lyukkártyaolvasó és -lyukasztó berendezések, sornyomtató stb.), valamint az ún. „külső tároló" egységeket (mágnesszalagos egységek, dob- és lemezmemóriák stb.). Újabban ez a megkülönböztetés egyre kevésbé indokolt, mert — mint látni fogjuk — mindezek az egységek programozástechnikai szempontból csaknem teljesen egyformán viselkednek.

Az első fogalom, amellyel a perifériákkal kapcsolatban meg kell ismerkednünk, a *rekord**. Ezen az információnak valamely olyan egységét értjük, amelyhez a periferiális átvitelek folyamán — azaz a periféria „egyetlen működése során" — mindig csakis *egészében* férhetünk hozzá. Egy rekord pl. a lyukkártyás átvitelek esetében egy kártya, a sornyomtatón egy sor, a lyukszalagon az olvasóberendezést vezérlő két „soremelés" (newline, vagy line feed) karakter közti rész információtartalma stb. Rekordok formájában tároljuk az információt a

* A rekord szónak jelenleg nincs elfogadott magyar megfelelője, így a szövegben kénytelenek vagyunk ezt használni.

mágnesszalagokon is. A rekord tehát az a legkisebb információmennyiség, amelyet egy adott periféria egyszerre át tud vinni (a beolvasást és a kiírást a továbbiakban közös néven *átvitelnek* nevezzük).

A rekord szemléletes fogalmából is kitűnik, hogy egy adott perifériális egység rekordjaihoz nem akármilyen sorrendben férhetünk hozzá: a *rekordoknak szigorúan kötött sorrendjük van.* Ezt pl. kártyaolvasó esetében a lyukkártyák sorrendje, sornyomtatón a kinyomtatott sorok egymásutánja szabja meg. Ahhoz, hogy a periféria k-adik rekordját átvihessük, előzőleg át kell vinnünk (esetleg átugratnunk) az első $k-1$ rekordot, a k-adik rekord átvitele után pedig csak a $k+1$-edik rekord van számunkra közvetlenül hozzáférhető helyzetben.

A rekordról kialakult szemléletes képünkhöz az is hozzá tartozik, hogy az átvitel sorrendje általában *nem megfordítható;* azaz ha a periferiális átvitel során egyszer eljutottunk egy adott rekordig, akkor az őt megelőző rekordokra nem térhetünk többé vissza. A lyukkártyaolvasó berendezés pl. az elolvasott kártyákat bedobja egy gyűjtőbe, ahonnan nem tudja újra elővenni. Hasonlóan nem tudja visszahúzni a lyukszalagolvasó az egyszer már beolvasott lyukszalag részt. Ez a tulajdonság azonban már nem bármely perifériális egységre jellemző, mert például a mágnesszalagos egységek képesek visszacsévélési (visszalépési) művelet végzésére, és így az egyszer már átvitt vagy átugratott rekordok újra hozzáférhetővé tehetők. Ennek megfelelően a FORTRAN nyelv tartalmaz visszalépési utasításokat is, ezeket azonban nyilván csak olyan perifériákra szabad kiadni, amelyek azt képesek végrehajtani.

Az olyan információtároló berendezéseket, amelyeken az egyes rekordok csak meghatározott sorrendben egymás után, vagy esetleg fordított sorrendben is elérhetők, *soros hozzáférésű* információhordozónak, soros hozzáférésű tárolónak nevezzük. A FORTRAN nyelv szempontjából minden periferiális egység soros hozzáférésű. Megjegyezzük, hogy a nagy elektronikus adatfeldolgozó rendszereknek általában vannak ún. *kötetlen hozzáférésű* külső tárolói is, amelyekre az jellemző, hogy minden bennük tárolt információ — sorrendi kötöttség nélkül — bármely időpillanatban egyenlő mértékben elérhető. Ebbe a kategóriába tartoznak pl. a mágneslemez (disk) és mágnesdob memóriák. A kötetlen hozzáférésű külső

tárolók — jellegüket tekintve — a központi memóriára emlékeztetnek, ui. általában számozott (címezett) memóriarekeszekre vannak felosztva, és az egyes rekeszek tartalma a címre való hivatkozás segítségével érhető el. A FORTRAN nyelv ilyen tárolók kezelésére nem ad direkt apparátust; ezt úgy kell érteni, hogy ha vannak egy gépnek kötetlen hozzáférésű külső tárolói, a FORTRAN felhasználó számára ezek is soros hozzáférésűek, az információt valóságos műszaki tulajdonságaiktól függetlenül — esetleg definíciószerűen meghatározott — rekordokban tárolják. Tehát ebben az esetben a gép programrendszere szimulálja a programozó számára a soros elérésű tároló tulajdonságait. (Megjegyezzük, hogy ez nem feltétlenül előny.)

A perifériális átviteli utasításokat minden esetben — egy vagy több — teljes rekordra adjuk ki. Egy rekord általában (de nem feltétlenül) egynél több *adatot* tartalmaz. Egy lyukkártyáról pl. nemcsak egy, hanem több változó értékét is beolvashatjuk, a sornyomtató egy sorában pedig egynél több számot is kiírathatunk.

A FORTRAN nyelv szempontjából az információ tárolásának két formáját különböztetjük meg. Az egyiket *karakterformának*, a másikat *bináris formának* nevezzük.*

A karakterforma az információnak az a megjelenési módja, ahogyan az a gép külső egységein (lyukkártya, lyukszalag, sornyomtató) megjelenik. A karakterformájú információ alfanumerikus jelekből (vagyis betűk, számok, írásjelek és vezérlő karakterek) áll, és vagy közvetlenül „olvasható" (pl. a sornyomtatón), vagy „leírató". Ezt a szemléletes képet ki kell azonban egészítenünk azzal, hogy karakterformájú információt tartalmazhatnak a gép külső tárolói (főként a mágnesszalagok), de a központi memória is. Ilyenkor az egyes karakterek (betűk, számok stb.) bizonyos bitkombinációkkal vannak kódolva. Minden számológépnek van egy ún. *belső karakterkódja*, amely azt határozza meg, hogy az illető gép jelkészletében létező karaktereket milyen módon ábrázoljuk nullák és egyesek segítségével. Egy karaktert a belső kódban általában 6—8 bitből álló jelsorozatokkal ábrázolunk. A karakter-

* A bináris forma angol neve „unformatted", a karakterformáé „formatted". A magyar szaknyelvben az előbbit „kötetlen formátumú" vagy „formázatlan", az utóbbit pedig „kötött formátumú" vagy „formázott" információábrázolásnak szokták nevezni. Ezeket az elnevezéseket nem tartjuk szerencsésnek, mert nem fejezik ki eléggé a fogalom lényegét.

formájú információt szokás még *BCD* (Binary Coded Decimal, binárisan kódolt decimális) információnak is nevezni.

A karakterformával szemben megkülönböztetjük a bináris ábrázolásmódot. Ez elsősorban a központi memória ábrázolási rendszere, de megjelenhet a külső tárolókon is. A bináris információ nem karakterek egymás mellé állított kódjaiból áll, hanem olyan bináris jelsorozatokból, amelyeken a gép közvetlenül el tudja végezni az utasításrendszerében meghatározott műveleteket. (A számadatok az aritmetika által előírt formában ábrázoltak, a negatív számok a gépre jellemző — pl. komplementáris — ábrázolásban s í. t.) A bináris ábrázolásmódban tárolt információ tehát — a karakterformájúval ellentétben — minden előzetes átalakítás (konverzió) nélkül felhasználható további számításokra. Éppen ezért a további számítások céljaira megőrzött számadatokat (közbülső és részeredmények) tároljuk ily módon, míg a karakterforma a be- és kimenő adatok ábrázolási módja. A FORTRAN nyelvben vannak utasítások mind bináris, mind karakterformájú információ átvitelére.

Karakterformájú, röviden karakterinformáció átvitele esetén az átvitt rekord mindig karakterek — fix vagy változó hosszúságú — sorozata. A karaktereket 1-től folytatólagosan megszámozzuk. Beszélhetünk egy rekord *karakterpozícióiról;* ezen az egyes karakterek „helyét" értjük a rekordon. Ha pl. a rekord 80 oszlopos lyukkártya, akkor az egyes karakterpozíciók a lyukkártya egy-egy oszlopának felelnek meg. A fix hosszúságú rekordokkal operáló tárolóegységekben minden rekordnak ugyanannyi karakterpozíciója van (pl. lyukkártya), a változó rekordhosszúságú tárolókban ez a szám változhat (lyukszalag). A kettő között előfordul egy átmenet is, mégpedig a maximált rekordhossz; itt a rekordoknak nem kell meghatározott hosszúságúaknak lenniük, de a karakterpozíciók száma nem haladhat meg egy előírt korlátot. Bizonyos gépi reprezentációkban ilyenek a mágnesszalagos egységek. Hangsúlyozzuk, hogy a rekordhosszt az illető gép vagy perifériális egység sajátságai szabják meg, nem pedig a FORTRAN nyelv definíciói.

Minden karakterinformáció átviteli utasításhoz meg kell adnunk bizonyos kísérő információt, amely beviteli utasítások esetén a rekord „szétosztásáról", kiviteli utasítások esetén pedig

a rekord „összeállításáról" rendelkezik. Lényegében ez a kísérő információ határozza meg a kapcsolatot a beolvasandó (kiíratandó) programbeli változók, és az input (output) rekord karakterpozíciói között; másként szólva, ez adja meg a belső (bináris) és a külső (karakter) ábrázolás közti konverziós műveletekre vonatkozó utasításokat. Ezek az utasítások többek közt arra vonatkoznak, hogy hány tizedesjegyre kell kerekíteni a gép belsejében tárolt bináris végeredményt; milyen fejléceket és magyarázó felíratokat kell elhelyezni az eredménylapon; hol, és milyen formában találja a gép a beolvasandó számokat az input lyukkártyán; hol van olyan rész a szalagon, amelyet ki lehet hagyni stb. Mindez a segédinformáció összességében meghatározza a rekord formátumát és egyes részei specifikálják a karakterformáról a bináris formára vagy visszafelé való áttérés módozatait. A FORTRAN nyelvben a segédinformációt az ún. FORMAT utasítás tartalmazza, melynek szoros kapcsolata, kölcsönhatása van a megfelelő READ (beolvasás), ill. WRITE (kiírás) utasításokkal. Ennek a kölcsönhatásnak a szabályait az 5. pontban tárgyaljuk majd.

Bár a fentiekben általánosságban elmondottak a mágnesszalagos külső tárolóegységekre is vonatkoznak, ez utóbbiakkal röviden külön is foglalkozunk. Ezt az indokolja, hogy a mágnesszalagos egységeknek néhány olyan tulajdonsága van, amelyek egyéb külső tárolóknál és perifériáknál nincsenek meg. Ilyen pl. a már említett visszalépési és újrabeolvasási lehetőség. A mágnesszalagos egységekkel kapcsolatban — de más külső tárolók esetében is — értelmezni szokták az információnak egy, a rekordnál magasabb egységét, melynek neve *file**. A file-t (akárcsak a rekordot) nehéz lenne szabatosan definiálni; ezen a fogalmon valamilyen értelemben összetartozó adatoknak több rekordból álló összességét értik. Szabatos definíció helyett inkább szemléletesen azt mondjuk, hogy file pl. egy mágnesszalag tekercs, amelyen valamely program adatai vannak tárolva, vagy a mágneslemez memóriának egy adott program számára fenntartott része. A file azonosítására általában egy file-név, továbbá néhány egyéb adat szolgál, melyeknek célja,

* Ejtsd: fájl. Eredeti jelentése dosszié, iratgyűjtő. Elfogadott magyar megfelelője jelenleg nincs, ezért a szövegben kénytelenek vagyunk ezt használni. Sajnos, a helyesírás sem következetes, mert a rekordot magyarosan, a file-t pedig angolosan szokták írni. A továbbiakban mi is ehhez alkalmazkodunk.

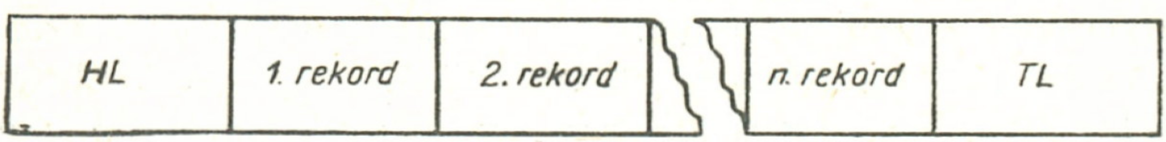
hogy kétséget kizáróan biztosítsák a file megkülönböztethetőségét minden más file-től. Ennek jelentőségét megértjük, ha meggondoljuk, hogy nagyobb számolóközpontokban több száz, sőt több ezer mágnesszalag tekercset is tárolhatnak, és ezeknek pontosan és félreérthetetlenül azonosíthatóknak kell lenniük.

A file azonosítását biztosító információ mágnesszalagok esetében a file legelső rekordjában van elhelyezve. Ezt a rekordot — amely tehát nem adatokat hordoz, hanem csupán azonosítási célokra van fenntartva — *header label**-nek nevezzük. Az ide írandó információ megadásának módját, valamint a megadandó információ pontos tartalmát az egyes gépi reprezentációk kézikönyvei írják le; ezekre nézve az ASA FORTRAN nem tesz előírásokat.

Ugyancsak különleges szerepe van a file utolsó rekordjának, melynek neve *trailer label***. Ez a file végét jelzi, vagyis azt a körülményt, hogy a file-ban nincs több információs rekord. A trailer label írására külön FORTRAN utasítás van (ENDFILE — lásd a 3. pontot). A FORTRAN programokat úgy kell megszerkeszteni, hogy mielőtt a program futása bármely okból megáll vagy megszakad, az írásra használt mágnesszalagokra trailer label kerüljön. Ennek hiányában a felírt információ visszaolvasásakor hibajelzés léphet fel. A trailer label írásának műveletét a file *lezárásának* nevezzük.

A 14. ábra a mágnesszalag rekordjainak sematikus elrendezését mutatja. Hangsúlyozzuk, hogy itt az ASA FORTRAN szempontjából tekintett elrendezésről van szó; hogy egyes gépi reprezentációkon ez az elrendezés miként realizálódik, a szakleírások körébe vágó kérdés.

Végezetül megjegyezzük, hogy a trailer label megfelelője egyes egyéb információhordozókon is előfordul. Lyukkártya ese-



14. ábra

* Ejtsd: heder lébel. Szokták magyarul fejcímkének nevezni, de ez az elnevezés nem terjedt el szélesebb körben.

** Ejtsd: tréler lébel. Szokták „end of file" rekordnak is nevezni. Elterjedt magyar megfelelője nincs.

tében pl. lezáró rekordnak szokás tekinteni az olyan lyukkártyát, amelynek első négy oszlopában egy-egy csillag karakter van, a többi karakterpozíciója pedig üres. Egyes gépi reprezentációs nyelvek definiálnak lezáró rekordot lyukszalagon is. Akárcsak a kártya esetében, itt is az „end of file" rekord bizonyos meghatározott karakterek egymásutánjából áll. A gép beolvasás után megvizsgálja, hogy a beolvasott rekord információs vagy end of file rekord-e, és ha programozási hiba folytán információs rekord helyett end of file rekordot talál, akkor hibajelzést generál.

2. A READ és WRITE utasítások szintaxisa és szemantikája

Az ASA FORTRAN mindössze két be- és kiviteli utasítást definiál, mégpedig egyet a beolvasásra (READ), egyet pedig a kiírásra (WRITE). Az ASA FORTRAN lényeges vonása, hogy *bármilyen* perifériális egységen történjék is az átvitel, annak vezérlését ugyanazok az utasítások végzik. Ez azt jelenti, hogy akár nyomtatjuk egy program eredményeit, akár szalagra lyukasztjuk, a kiíró utasítás mindenképpen a WRITE. Ez lényeges könnyítést jelent abban az esetben, ha egy programot át kell alakítani egyik perifériatípus használatáról egy másik típusú periféria használatára.

Ez a tulajdonsága azonban nem minden FORTRAN változatnak van meg; bizonyos változatokban pl. a nyomtatást a PRINT, a kártyalyukasztást a PUNCH, a mágnesszalagról történő be-, ill. kivitelt READ INPUT TAPE, ill. WRITE OUTPUT TAPE utasítások jelölik, és az itt felsoroltakon kívül további be- és kiviteli utasítások is definiálva vannak.

Szintaktikus szempontból a beolvasási és kiírási utasítások a következő alakúak:

$$\text{READ}\quad (u,f)\quad \text{L} \tag{6.1}$$

$$\text{READ}\quad (u)\quad \text{L} \tag{6.2}$$

$$\text{WRITE}\quad (u,f)\quad \text{L} \tag{6.3}$$

$$\text{WRITE}\quad (u)\quad \text{L} \tag{6.4}$$

A fentiekben *u* és *f* előjeltelen egész számok lehetnek, L pedig az ún. I/O *lista**. Ez utóbbi szintaxisára később térünk vissza. *u* lehet index nélküli egész változó is.

Bizonyos FORTRAN változatokban az *u*-t, ill. az *u*, *f* számpárt nem kell zárójelbe tenni.

A fenti utasítások közül (6.1) és (6.3) karakterátvitelt, (6.2) és (6.4) pedig bináris átvitelt jelent. Ennek a pontnak a hátralevő részében főként az I/O lista szintaxisával és szemantikájával foglalkozunk, és ebből a szempontból nem lényeges, hogy bináris, vagy karakterátvitelről van-e szó, és hogy az „átvitel" beolvasást, vagy kiírást jelent-e. Mielőtt azonban rátérnénk az I/O lista részletes tárgyalására, röviden beszélnünk kell az utasításokban szereplő *u*, ill. *f* számokról.

Az *f* számról itt csak annyit, hogy ez az átviteli utasításhoz rendelt FORMAT utasítás címkéje. Mint már az előző pontban említettük, minden karakterátviteli utasításhoz hozzá van rendelve egy FORMAT utasítás, amely a binárisról a karakterformára vagy visszafelé való áttérést szabályozza (részletekre nézve l. a 4. pontot). Az 5. pontban látni fogjuk, hogy *f* lehet tömbazonosító is.

Az átviteli utasításokban szereplő *u* mennyiség az ún. *logikai perifériaszám*. Ennek jelentése a következő. Láttuk, hogy az ASA FORTRAN-ban mindössze kétféle I/O utasítás létezik (READ és WRITE), tehát az utasítást meghatározó alapszóból nem derül ki, hogy az átviteli utasítás melyik perifériális egységre vonatkozik. Ezt az információt adja meg a programozó a logikai perifériaszám segítségével, amikor minden egyes perifériális egységhez hozzárendel egy számot, és ezzel rögzíti, hogy az adott programban a sornyomtatót pl. 2-es, a kártyaolvasót 4-es, a felhasznált két mágnesszalagos egységet pedig 11-es és 12-es sorszámokkal fogja jelölni. Ebben az esetben a sornyomtató logikai perifériaszáma 2-es, a kártyaolvasóé 4-es, a mágnesszalagos egységeké pedig 11-es, ill. 12-es. A programban ezután a nyomtatási utasításokat mindig a 2-es logikai perifériára adjuk ki (azaz $u=2$), a beolvasásokat $u=4$ jelenti

* Az I/O az angol Input—Output (be- és kivitel) szópár rövidítése; jelölésként a magyar szakirodalomban is elterjedt, ezért a továbbiakban mi is használjuk. Hasonlóan, bevitel esetén szokás beszélni „input rekord"-ról, kiírás esetén pedig „output rekord"-ról.

s í. t. A számok megválasztásában a programozó nincsen korlátozva (bár egyes gépi reprezentációkban maximálják a kiosztható legnagyobb logikai perifériaszámot pl. $u=2048$-ban).

Azt várnánk, hogy az ASA FORTRAN tartalmaz valamiféle deklarációs utasításokat, amelyek segítségével deklarálni lehetne a logikai perifériaszámok és a konkrét perifériális egységek egymáshoz rendelését. Ilyen utasításokat azonban az ASA FORTRAN-ban nem definiálnak; ezek az egyes gépi reprezentációs nyelvekben vannak, pontos használatukat a kézikönyvek írják le. Ha egy FORTRAN programot, amelyet eredetileg pl. sornyomtatón megjelenő outputra terveztek, át kívánunk írni szalaglyukasztóra, nem kell egyebet tennünk, mint a gépi reprezentáció periféria hozzárendelési utasításait megfelelő másik utasításokkal helyettesítenünk.

Ezek után áttérünk az I/O listák szintaxisának és szemantikájának tárgyalására.

Formailag az L lista vagy egyetlen listaelemből, vagy listaelemek egymástól vesszővel elválasztott sorozatából áll, vagy pedig üres. A listaelemek kétfélék lehetnek, mégpedig: *egyszerű listaelemek* és belső ciklusutasítást tartalmazó — a továbbiakban *összetett — listaelemek.*

Egyszerű listaelemek lehetnek:

1. Egyszerű változók;
2. Indexes változók;
3. Tömbazonosítók.

Egyetlen magában álló egyszerű listaelem, vagy egyszerű listaelemek egymástól vesszővel elválasztott sorozata alkothat egy ún. *egyszerű* I/O *listát.* Tehát egyszerű listának minden eleme egyszerű listaelem.

Példák egyszerű I/O listára:

A, B, Q(4,3), S, R(I,J) (6.5)

G (6.6)

(6.5) többelemű lista, (6.6) egyelemű lista. A szereplő A, B, S és G azonosítók akár skaláris, akár indexes változók (tömbök) azonosítói lehetnek.

Összetett I/O *listaelemhez* oly módon juthatunk, hogy egy *tetszőleges* (tehát nem okvetlenül *egyszerű*) I/O lista után vesszőt teszünk, ezt követőleg egy ún. *belső ciklus specifikációt* helyezünk el, majd az így kapott jelsorozatot zárójelbe zárjuk.

A belső ciklus specifikáció alakja a következő:

$$\mathsf{I} = m_1, m_2, m_3 \tag{6.7}$$

vagy

$$\mathsf{I} = m_1, m_2 \tag{6.8}$$

ahol I index nélküli egész változó, m_1, m_2, m_3 pedig pozitív egész szám, vagy pozitív egész értékkel bíró egész változó lehet (vö. Harmadik fejezet 5. pont, DO utasítás). Példák összetett listaelemre:

$$\text{(A, B Q(4,3), S, R(I,J), J=1, 21, 2)} \tag{6.9}$$

$$\text{(U(I), V(I), W(I), I=1, N)} \tag{6.10}$$

Mint látható, a (6.9) összetett listaelemet oly módon állítottuk elő a (6.5) egyszerű listából, hogy hozzáillesztettük a

$$\text{J=1, 21, 2}$$

(6.7) típusú belső ciklus specifikációt, és a kapott jelsorozatot zárójelbe zártuk. Hasonlóan, a (6.10) összetett listaelem egy háromelemű egyszerű listából, és egy hozzáillesztett (6.8) típusú belső ciklus specifikációból áll.

Az összetett I/O listaelemek ugyanúgy használhatók I/O listák felépítésére, mint az egyszerűek. Az olyan listát, amelynek van összetett eleme, *összetett* I/O *listának* nevezzük. Példák összetett I/O listákra:

$$\text{(A(I), I=1,M), (B(I), I=1,M)} \tag{6.11}$$

$$\text{X, Y, (G(K), K=I,J,2)} \tag{6.12}$$

A (6.11) lista két összetett elemből, a (6.12) lista pedig két egyszerű, és egy összetett elemből áll.

Ezzel lényegében felsoroltuk az I/O listák megszerkesztésével kapcsolatos szintaktikus szabályokat, azonban az eddig elmondottak további, összetettebb kombinációk létrehozását is lehetővé teszik.

Az összetett listaelem definíciójakor azt mondottuk, hogy képzésében nem okvetlenül egyszerű listából kell kiindulnunk. A (6.9) és (6.10) példákban szereplő összetett elemek viszont egyszerű listákon alapultak. Nézzük meg most, milyen jelsorozat jön létre, ha egy összetett listaelem előállításához összetett listából indulunk ki?

Tekintsük pl. a (6.9) összetett listaelemet, amelyet felfoghatunk egyelemű összetett listaként is. Hajtsuk végre az összetett listaelem előállítására leírt eljárást: tegyünk a lista után vesszőt, illesszünk hozzá egy belső ciklus specifikációt, majd zárjuk az így kapott jelsorozatot zárójelbe! Az eredmény egy „kétszeresen összetett" listaelem:

((A, B, Q(4,3), S, R(I,J), J = 1, 21,2), I = 1, 22, 7) (6.13)

A szintaxisból nyilvánvaló, hogy ezt az eljárást akárhányszoros mélységben tovább lehetne folytatni.

Hasonlóan képezhetünk összetett listaelemet a (6.11) összetett listából:

((A(I), I = 1,M), (B(I), I = 1,M), K = 1,5) (6.14)

Végül egy harmadik példa, melyhez hasonlók a gyakorlati programozásban igen gyakran fordulnak elő:

(K, (J, (I, (A(I,J,K)), I = 1, N), J = 1, M), K = 1,L) (6.15)

A két utóbbi példa elemzését az olvasóra bízzuk.

Befejezésül még egy rövid szintaktikai megjegyzés: az I/O listák szintaxisa (ellentétben pl. az aritmetikai kifejezések szintaxisával) nem tűri a redundáns zárójelek alkalmazását. Ez alól egyetlen kivétel van: *nem számít szintaktikus hibának, ha az egyszerű* I/O *listákat egyetlen (de csakis egyetlen) redundáns zárójelpárral vesszük körül.* Tehát szintaktikusan helyes pl. a következő READ utasítás:

READ (5)(A, B, C) (6.16)

viszont már szintaktikusan hibás

READ (5)((A, B, C)) (6.17)

Áttérünk most az I/O listák szemantikájának tárgyalására.

Az I/O listák mindig be- és kiviteli utasítások részei, mégpedig kijelölik az átvitelben résztvevő számadatokat (amelyek mindig bizonyos programbeli változók aktuális értékei), beleértve a sorrendiséget is.

Az I/O listák pontos értelmezését a következő szabályok adják:

1. szabály: A lista értelmezése balról jobbra tartó sorrendben történik („balról jobbra szabály").

2. szabály: A listán szereplő skaláris és indexes változók azonosítói a megfelelő változó aktuális értékének átvitelét jelentik. Ezt úgy kell érteni, hogy (1) kiírási (WRITE) művelet esetén a változó aktuális értékét írja ki az output periférián (pl. sornyomtatón), (2) beviteli (READ) művelet esetén pedig a beolvasás megtörténte után a változó aktuális értékként felveszi az input periférián soronkövetkező számértéket. Kiírási műveletek esetén a lista elemeinek aktuális értékei változatlanok maradnak.

3. szabály: A listán szereplő tömbazonosítók a megfelelő *tömb valamennyi elemének átvitelét* jelentik, mégpedig a címfüggvény által meghatározott sorrendben (lásd a Negyedik fejezet 3*d* pontját). Az egyes tömbelemek aktuális értékének átvitelére nézve az előző bekezdésben elmondottak irányadók*.

Példák

1. Tekintsük a következő kiírási utasítást:

WRITE (5) F, G, A (6.18)

ahol F és G skaláris változók, A pedig 3×3-as tömb. E feltevés mellett (6.18) összesen 11 szám kiírását jelenti. Az output periférián az aktuális értékek a következő sorrendben jelennek meg: F, G, A(1,1), A(2,1), A(3,1), A(1,2), A(2,2), A(3,2), A(1,3), A(2,3), A(3,3). A kiírás után valamennyi változó aktuális értéke változatlan marad.

2. Legyenek R és Q mindketten 2×2-es tömbök. Ekkor a

READ (4) R, Q (6.19)

beolvasási utasítás összesen 8 szám beolvasását írja elő. Az input periférián soronkövetkező nyolc szám beolvasódik az R(1,1), R(2,1), R(1,2), R(2,2), Q(1,1), Q(2,1), Q(1,2), Q(2,2) tömbelemekbe. Már e helyen nem árt felhívni a figyelmet arra, hogy bármilyen beolvasási műveletről legyen is szó, a bemenő adatokat úgy kell előkészíteni a program számára, hogy azok az input információhordozón a megfelelő sorrendben kövessék egymást (példánkban előbb az R tömb, majd a Q tömb elemeit kell felvinnünk, mindkettőt oszlopfolytonosan).

* Van egy negyedik szabály is, amely az összetett I/O listaelemekre vonatkozik. Ezt a 204. oldalon adjuk.

Az összetett I/O-listák szemantikájának tisztázása céljából először az összetett I/O listaelemek szemantikájával foglalkozunk.

Mint már láttuk, az összetett I/O listaelem formálisan a következő alakú:

$$(\mathsf{L}, \mathsf{I} = m_1, m_2, m_3) \tag{6.20}$$

illetőleg

$$(\mathsf{L}, \mathsf{I} = m_1, m_2) \tag{6.21}$$

(lásd pl. (6.10)), ahol L tetszőleges I/O lista. Az összetett listaelemek *csoportos átvitel* végrehajtására szóló utasítást jelentenek, a következők szerint:

1. Az I változó — a továbbiakban *ciklusváltozó* — felveszi az m_1 kezdőértéket;

2. a ciklusváltozó így kapott aktuális értékével átvitelre kerül az L I/O lista valamennyi eleme;

3. I értéke megnövelődik az m_3 növekményparaméter értékével (ha a listaelemben szereplő belső ciklus specifikáció (6.8) alakú, akkor 1-gyel);

4. megvizsgálódik, hogy I meghaladta-e már az m_2 végérték paraméterértékét;

5. ha I értéke még nem haladta meg ezt az értéket, akkor ismételten végrehajtódnak a 2.—5. alattiak;

6. ha I már meghaladta a végérték paraméter által előírt értéket, akkor az összetett I/O listaelem értelmezése véget ért.

A fentiekből kitűnik, hogy az összetett I/O listaelemek értelmezése teljesen analóg a DO utasítások szemantikájával (vö. Harmadik fejezet 5*a* pont).

Példák

1. Tekintsük a (6.10) összetett cikluslista-elemet:

$$(\mathsf{U(I)}, \mathsf{V(I)}, \mathsf{W(I)}, \quad \mathsf{I} = 1, \mathsf{N})$$

Ennek értelmezése a következő:

1. Az I ciklusváltozó felveszi az m_1 értéket, esetünkben 1-et.

2. Ezen aktuális érték mellett átvitelre kerülnek az I/O lista elemei, nevezetesen U(1), V(1) és W(1).

3. Minthogy példánkban a belső ciklus specifikáció (6.8) típusú, a ciklusváltozó értéke 1-gyel növelődik.

4. Megvizsgálódik, hogy I értéke meghaladta-e már az N-et (feltesszük pl., hogy N = 5).

5. Mivel még nem haladta meg, visszatérés a 2. pontra; ennek végrehajtása ezúttal az U(2), V(2) és W(2) elemek átvitelét jelenti.

6. Az átvitel addig folytatódik, míg I el nem érte az N = 5 értéket (az átvitel még ezzel az értékkel is végrehajtódik).

Végeredményben — feltéve, hogy N = 5 —, a (6.10) összetett cikluslista-elem az U, V és W vektorok elemeinek a következő sorrendű átvitelét jelenti: U(1), V(1), W(1), U(2), V(2), W(2), U(3), V(3), W(3), U(4), V(4), W(4), U(5), V(5), W(5).

2. Tekintsük (6.9)-et, és legyen I aktuális értéke 8. Ezen feltevés alapján a listaelem értelmezése a következő:

1. J felveszi a kezdőértéket, 1-et.

2. Átvitelre kerülnek az I/O lista megfelelő elemei: A, B, Q(4,3), S és R(8,1).

3. J értéke összehasonlítódik az előírt végértékkel, 21-gyel. Az összehasonlítás eredménye negatív; J a végértéket még nem haladta meg.

4. J értéke megnövelődik a növekményparaméter értékével, 2-vel. Adódik J = 3.

5. Átvitelre kerülnek a J = 3-nak megfelelő listaelemek. Ezek: A, B, Q(4,3), S és R(8,3).

6. A leírt eljárásnak a fentiek szerinti ismétlésével a (6.10) I/O listaelem az átvitel következő sorrendjét írja elő:

A, B, Q(4,3), S, R(8,1),

A, B, Q(4,3), S, R(8,3),

A, B, Q(4,3), S, R(8,5), (6.22)

. .

A, B, Q(4,3), S, R(8,21).

(6.22)-őt közelebbről megvizsgálva észrevesszük, hogy az I/O listának azok az elemei, amelyek nem függnek a J ciklusválto-

zótól, többször is átvitelre kerülnek, míg azok, amelyek függnek a ciklusváltozótól, csak egyszer. Ezért általában a belső ciklust elsősorban olyan listaelemekkel kapcsolatban alkalmazzuk, amelyek függnek a ciklusváltozótól, de ha célunk ugyanannak az elemnek többszöri átvitele, akkor ettől el is térhetünk.

3. Tekintsük a következő I/O listaelemet, mely az első példában vizsgált (6.10) elem egészen csekély módosításával áll elő:

$$(I, U(I), V(I), W(I), \quad I=1, N) \qquad (6.23)$$

Az eltérés, mint látjuk, abban van, hogy (6.23)-ban az összetett listaelemben szereplő I/O lista tartalmazza a ciklusváltozót. Ezt *kiviteli utasítások esetén megengedi* az ASA FORTRAN, *beolvasási utasításokra* viszont *megtiltja.*

Az ismertetett szemantikai szabályok alkalmazásával — ismét feltéve, hogy N = 5 —, a következő kiírási sorrend adódik: 1, U(1), V(1), W(1), 2, U(2), V(2), W(2), 3, U(3), V(3), W(3), 4, U(4), V(4), W(4), 5, U(5), V(5), W(5). Az output rekordon tehát minden egyes kiírási ciklusban megjelenik a ciklusváltozó aktuális értéke is. Ezt a fogást felhasználhatjuk pl. a sornyomtatón kiíratott számanyag sorainak megszámozására.

Visszatérünk most arra a kérdésre, hogy miért nem lehet (6.23) egy READ listának is eleme? Erre a következő szemantikai szabály ad választ: *az összetett* I/O *listaelemeknek olyannak kell lenniük, hogy a végrehajtott átvitelek sem a ciklusváltozónak, sem pedig a kezdőérték-, növekmény- és végértékparamétereknek az értékét ne változtassák meg.* A már ismert szemantikai szabályok szerint a kiírási jellegű átvitelek nem változtatják meg az átvitt változók értékét, tehát ez a szabály csak a READ utasításokra jelent korlátozást. (6.23) ennek nem tesz eleget, minthogy az input perifériáról beolvasott számadatok megváltoztatnák I értékét.

Az egyszerű I/O listák, valamint az összetett I/O listaelemek értelmezésével kapcsolatos szabályok ismeretében most már könnyen elintézhetjük az összetett listák szemantikáját is. A már ismert szabályokhoz (199—200 old.) ugyanis csupán egyet-

len újabbat kell hozzátennünk (a 200. oldali számozás szerint ez a szabály a 4-es sorszámot viseli):

4. szabály: A listán előforduló összetett listaelemek az elem által meghatározott összes számadat átvitelét jelentik, az idevonatkozó szemantikai szabályok (lásd 201. old.) szerinti sorrendben.

Mint az előzőkben láttuk, a (6.20) és (6.21) összetett listaelemek részként tartalmazhatnak összetett I/O listákat. Magától értetődik, hogy ezeket a „belső" listákat is az imént megismert szabályok szerint kell átvinni; vagyis azok a szabályok, amelyek az „egészre" vonatkoznak, egyszersmind érvényesek annak egy részére is. A matematikában az ilyen szabályokat *rekurzívnek* szokás nevezni, és azt is mondhatjuk, hogy az összetett listaelem ebben az értelemben *rekurzív fogalom.* Hasonlóan rekurzív fogalom pl. az aritmetikai kifejezés, melynek egy zárójelekkel elhatárolt része ismét aritmetikai kifejezés, és a kiszámítási szabályok is rekurzívek, amennyiben a zárójelekkel elhatárolt részt ugyanúgy kell kiszámítani, mint magát a teljes kifejezést.

Példák

1. Legyenek A és B 3×3-as tömbök, és tekintsük a következő — egyelemű, összetett — I/O listát:

$$((A(I,J),\ B(I,J),\ I=1,3),\ J=1,3) \qquad (6.24)$$

A 201. oldalon leírtak szerint, legelőször J felveszi az 1 értéket. Ezután át kell vinnünk az I/O lista elemeit. Ez jelen esetben maga is egyelemű összetett lista, melynek átvitele — figyelembe véve, hogy J-t most rögzítettük — a következő tömbelemek átvitelét jelenti (sorrendben): A(1,1), B(1,1), A(2,1), B(2,1), A(3,1), B(3,1). Ezután, megnövelve J értékét, a következő tömbelemek kerülnek átvitelre: A(1,2), B(1,2), A(2,2), B(2,2), A(3,2), B(3,2); végül J=3-mal A(1,3), B(1,3), A(2,3), B(2,3), A(3,3), B(3,3). Figyeljünk arra, hogy ez nem ugyanaz, mintha az I/O lista egyszerűen így festene:

A, B

2. Írjunk I/O listát a 10×10-es G tömb elemeinek sorfolytonos átvitelére!

Megoldás:

$$((G(I,J), \quad J=1, 10), \quad I=1, 10) \tag{6.25}$$

Ez ismét nem egyenértékű az egyetlen G elemből álló I/O listával.

Végezetül még megjegyezzük, hogy a DO utasításokhoz hasonlóan a többszörösen összetett I/O listák esetében is a *legbelső* ciklus kerül *legtöbbször* végrehajtásra, más szóval a legbelső ciklusváltozó változik a „leggyorsabban". Ez természetesen nem új szemantikai szabály, hanem az előzők összefoglalása, a gyakorlati programozás szempontjából azonban hasznos, mert tömören kifejezi az eddig elmondottak lényegét.

3. Bináris formájú átvitelek. Mágnesszalagkezelő utasítások: BACKSPACE, REWIND, ENDFILE

A bináris formájú átvitelek utasításainak alakját az előző 2. pontban, a (6.2) és (6.4) képletekkel megadtuk. A tárgyalt értelmezési szabályok alapján már csaknem teljes egészében ismerjük a bináris átvitelek szemantikáját is. Mindössze egyetlen nyitott kérdés maradt, mégpedig az, hogy milyen kapcsolatban van az I/O lista és a periférián rögzített I/O rekord? Közelebbről, hogyan darabolódik fel az output lista output rekordokra, illetve hány rekordot kell beolvastatnunk egy input lista feldolgozása közben?

Erre a kérdésre roppant egyszerű a válasz: *bináris átvitelek esetén az I/O lista egyetlen rekord átvitelét jelenti.* Ezt másként úgy is megfogalmazhatjuk, hogy *a rekordhosszt az I/O lista határozza meg.* (Ez az egyszerű szabály a karakter formájú információátvitel esetében nem érvényes!)

Az iménti szabályt *kiírási* műveletekre úgy kell értenünk, hogy valahányszor a gép hozzákezd egy WRITE utasítás végrehajtásához, mindannyiszor új rekordot nyit, és az output lista által meghatározott valamennyi számadatot egyetlen rekordban helyezi el. A *bináris kiírás* tehát *nem darabolja fel az output listát több rekordra*, akárhány elem kiírása legyen is előírva.

Beolvasási műveletekre az említett szemantikai szabály azt jelenti, hogy *valamennyi beolvasandó számadatnak ugyanabban a rekordban kell elhelyezve lennie;* a bináris beolvasási művelet végrehajtásának megkezdésekor a gép új rekordot kezd, és további rekordokat a beolvasási művelet végrehajtása közben megkezdeni már nem lehet.

A rekord fogalmával kapcsolatban szükségesnek tartjuk megjegyezni, hogy a konkrét perifériális egységekre értelmezett rekord fogalom, és az ASA FORTRAN-ban értelmezett rekord fogalom között eltérés lehet, amennyiben a periferiális egységen megjelenő ún. *fizikai rekord* lehet nagyobb is, és kisebb is, mint egy FORTRAN nyelvbeli, ún. *logikai rekord.* A gép programozási rendszere ilyen esetben gondoskodik arról, hogy a logikai rekordot automatikusan szétbontsa több fizikai rekordra, illetve hogy több logikai rekordból összeállítson egy fizikai rekordot. Mi a továbbiakban rekordon mindig logikai rekordot értünk. Azt, hogy egy bizonyos konkrét rendszerben a logikai és a fizikai rekordok milyen kapcsolatban vannak egymással, a gépi reprezentáció kézikönyvei írják le.

Mindenfajta beolvasási művelettel kapcsolatban ügyelni kell arra, hogy az input rekord ne tartalmazzon kevesebb számadatot, mint amennyinek beolvasását az input lista előírja, vagyis hogy az input rekord ne legyen túl rövid. Több számadatot tartalmazhat az input rekord, de nem szabad megfeledkeznünk arról, hogy a rekord végén álló fölös számadatok a beolvasás során figyelmen kívül maradnak, és hogy rekord közepébe „belekapni" semmiféle READ utasítás sem képes.

A (6.1)-től a (6.4)-ig terjedő I/O utasítások szemantikai szabályait ki kell még egészítenünk az üres lista esetével. Az üres I/O listájú

1. bináris READ utasítás a soron következő rekord átugratását (kihagyását),

2. bináris WRITE utasítás egy üres rekord kiírását jelenti.

Karakter formájú információátvitelekre a fentiek nem vonatkoznak; a megfelelő szabályokat a 4. pontban tárgyaljuk.

Mielőtt a bináris átvitelre konkrét programozási példákat mutatnánk, röviden áttekintjük az ún. *szalagkezelő utasításokat*, amelyek nem információt visznek át, hanem a mágnesszalagnak a kívánt helyzetbe való beállítását vagy lezárását intézik. Az ezekben az utasításokban megadott logikai perifériaszámnak egy mágnesszalagos egységhez kell tartoznia.

A szalagkezelő utasítások a következők:

BACKSPACE *u* (6.26)

REWIND *u* (6.27)

ENDFILE *u* (6.28)

ahol *u* a mágnesszalagos egység logikai perifériaszáma. (Előjeltelen egész szám, vagy index nélküli egész változó lehet.)

A BACKSPACE utasítás a mágnesszalagon *egy rekorddal való visszalépést* jelent, hatására a mágnesszalag a megelőző rekord elé áll.

A REWIND utasítás a szalag *teljes visszacsévélését* idézi elő: végrehajtásának eredményeként a szalag első adatrekordja beolvasásra készen áll.

Az előző két utasításra az a közös szemantikai szabály érvényes, hogy ha a szalag az utasítás végrehajtása előtt már visszacsévélt állapotban volt, akkor mindkettő teljesen hatástalan.

A harmadik szalagkezelő utasítás, az ENDFILE, a szalagra egy *trailer label* (end of file rekord) *írását* eredményezi. A trailer label jelentésére röviden utaltunk az 1. pontban; a részletes tárgyalás a gépi reprezentációs nyelvek szakleírásainak körébe vág.

Egyes gépi reprezentációs nyelvek az ENDFILE utasítás használatát nemcsak mágnesszalagos egységekre engedélyezik, hanem pl. lyukszalag vagy lyukkártya output esetére is. Megjegyzendő, hogy az ASA FORTRAN jelentés sem kifejezetten mágnesszalagos egységről beszél, hanem „soros elérésű file"-ról, amelyen azonban hallgatólag mágnesszalagos egységet értenek.

E pont további részében gyakorlati programozási feladatokon mutatjuk be eddigi ismereteink konkrét alkalmazásait.

Példák

1. Legyen adva egy 50 bináris rekordból álló mágnesszalag, ahol a rekordok egyenként 250 szavasak. Írjunk FORTRAN programot ennek a mágnesszalagnak változtatás nélküli átmásolására! Az input mágnesszalag logikai perifériaszáma legyen 5-ös, az output szalagé 6-os! (Feltesszük, hogy a szalagon csupa valós szám van.)

A program:

```
      MASTER MTCOPY
      DIMENSION REK(250)
      DO 2 I=1, 50
      READ (5) REK
2     WRITE (6) REK
      ENDFILE 6
      REWIND 6
      REWIND 5
      STOP
      END
```

(6.29)

A fenti programban egy DO ciklus végzi a rekordok átmásolását: READ utasítással történő beolvasását, majd a másik szalagra való kiírását WRITE utasítással. A REK tömb egy 250 elemű munkamező, amely egy beolvasott input rekordot ideiglenesen tárol, míg az átmásolás meg nem történik és a következő rekordot be nem olvassák az előző helyébe. (6.29) kapcsán emlékeztetjük az olvasót arra a szemantikai szabályra, amely szerint az I/O lista meghatározza a rekordhosszt. Ez biztosítja, hogy a gép minden egyes ciklusmenetben egy új rekordot olvasson be, és egy új rekordot írjon ki. Minden kiírásra használt mágnesszalagot a munka befejeztével „le kell zárni", azaz trailer labelt kell ráírni; ezenkívül minden mágnesszalagot (akár kiírásra, akár beolvasásra használtuk) a munka végeztével vissza kell csévélni. Erre valók a program végén elhelyezett ENDFILE és REWIND utasítások.

Egyes gépi reprezentációk kiírására használt mágnesszalagokra a REWIND-ba beleértik az ENDFILE-t is, vagyis ha az output szalagot a munka végeztével visszacsévéltetjük, akkor nem szükséges külön ENDFILE utasítást írnunk. Az ENDFILE megléte azonban semmiképpen sem hiba.

2. Módosítsuk az előző feladatot annyiban, hogy ezúttal az input szalag fordított átmásolását írjuk elő! (Nemcsak az

egyes rekordokat kell fordított sorrendben átvinnünk, hanem a rekordokon belül is fordított sorrendben követeljük a számadatok átvitelét!)

A módosított program a következő:

```
   MASTER REVCOPY
   DIMENSION REK1(250), REK2(250)
   DO 1 I=1, 49
1  READ (5)
   DO 2 I=1, 50
   READ (5) REK1
   BACKSPACE 5
   BACKSPACE 5
   DO 3 K=1, 250                      (6.30)
   J = 251-K
3  REK2(K)=REK1(J)
2  WRITE (6) REK2
   ENDFILE 6
   REWIND 6
   STOP
   END
```

Ez már jóval bonyolultabb program az előzőnél. Az első DO ciklus előrelépést jelent az input szalag legutolsó rekordjáig, mivel egy üres listájú READ utasítás hatására egy rekord átugrása következik be (lásd az üres I/O listákról mondottakat!). A második DO ciklusban a tényleges beolvasást két BACKSPACE utasítás követi, amely biztosítja, hogy minden egyes rekord beolvasása után a megelőző rekord elé lépjünk. Ezt követőleg egy rövid belső DO ciklus az input rekordot, mely REK1-ben eredeti sorrendben van elhelyezve, fordított

sorrendben átrakja az output rekord helyére, a REK2 vektor elemeibe. Az egyes rekordokon végigfutó — a második DO — ciklus a REK2 vektor kiírásával végződik. Amikor ez a ciklus „lejárt", a „fordított" átmásolás is megtörtént. A program végén áll a kötelező ENDFILE utasítás, majd az output szalag visszacsévéléséről gondoskodó REWIND 6 utasítás. Az input szalagot ezúttal nem kell külön utasítással visszacsévéltetnünk, mert maga az algoritmus olyan, hogy befejeztével a szalag visszacsévélt állapotban marad (sőt, a legutoljára végrehajtott BACKSPACE utasítás az ismertetett szemantikai szabályok értelmében hatástalan, mert a szalag már ennek végrehajtása előtt teljesen visszacsévélt állapotban van).

Lényegesen egyszerűsíthető a (6.30) program minden olyan gépi reprezentációban, amely megengedi „nem standard" indexkifejezések használatát. Ha ugyanis megengedettek $c-\mathrm{I}$ alakú indexek, ahol c egész szám és I egész változó, akkor (6.30) a következőképpen módosítható:

```
      MASTER REVCOPY
      DIMENSION REK (250)
      DO 1 I=1, 49
    1 READ (5)
      DO 2 I=1, 50
      READ (5) REK
      BACKSPACE 5                                   (6.31)
      BACKSPACE 5
    2 WRITE (6)(REK(251-J), J=1, 250)
      ENDFILE 6
      REWIND 6
      STOP
      END
```

Az egyszerűsítés lényege, hogy a nem standard indexkifejezés lehetővé teszi a 2-es címkéjű utasításban egy összetett listaelem alkalmazását, így egy 250 elemű munkavektort, és egy teljes DO ciklust takarítunk meg.

A fejezet végén, a 3. feladatban e program kapcsán egy szemantikai problémára hívjuk fel az olvasó figyelmét.

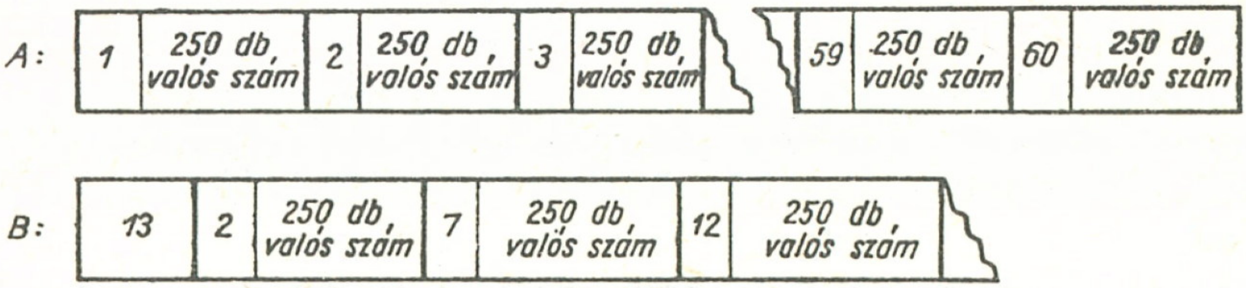
3. Legyen adva két mágnesszalagunk, jelölje ezeket *A*, ill. *B*! Az *A* mágnesszalag összesen 60 bináris rekordot tartalmaz, melyek mindegyike összesen 251 elemű. Minden rekord első számadata egy egész ábrázolású szám, amely a rekord sorszámát jelenti a szalagon. A többi 250 elem valós ábrázolású szám. A *B* mágnesszalag első rekordja egyelemű: egyetlen egész számot tartalmaz, amely a szalagon található rekordok számát jelenti. A további rekordok hasonló formátumúak az *A* szalag rekordjaihoz, de az első elem helyén álló számok nem feltétlenül egyesével növekszenek. Annyit felteszünk, hogy növekvő sorozatot alkotnak, és hogy 60-nál kisebbek.

A feladat egy harmadik, *C* mágnesszalag előállítása, az alábbiak szerint: az *A* mágnesszalagnak azokat a rekordjait, amelyekkel megegyező sorszámú rekord *nem található* a *B* mágnesszalagon, az *A* szalagról átmásoljuk *C*-re, azokat a rekordokat pedig, amelynek megfelelő sorszámú rekord *található* a *B* szalagon is, a *B* szalagról másoljuk át *C*-re.

Ez a feladat egy igen egyszerű, ún. „összefuttatási" program. Összefuttatási feladatokon olyan adatfeldolgozási feladatokat értünk, amelyekben több input file adataiból — bizonyos feldolgozás után — egy, vagy több output file-t kell előállítanunk. Ez nem precíz matematikai fogalom, és ezért nem időzünk most annál a lehetséges megjegyzésnél, mely szerint az előbbi meghatározás ráhúzható minden olyan mágnesszalagos inputtal dolgozó programra, amelynek egynél több input mágnesszalagja van. A pontos határt valóban nehéz megvonni, azonban általában akkor szokás összefuttatásról beszélni, ha a feladat leglényegesebb része az output rekordok összeállítása az input rekordok alapján, és az esetleges közbülső feldolgozás csak kisebb mértékű.

Feladatunk ezen tágabb feladatosztályon belül egy szűkebb, és ezért jobban körülhatárolható csoportba tartozik, nevezetesen az ún. „*file felújítási*" vagy „*file szerkesztési*" feladatok csoportjába. A file felújítás (szerkesztés) során egy input file-t valamilyen „javító file" segítségével „kijavítanak", azaz némiképpen módosított alakban átmásolnak. A módosítás lehet egyes rekordok megváltoztatása, újabb rekordok beszúrása, régiek elhagyása. Példánkban a kijavítandó file az *A* mágnesszalag, a javító file *B*, a kijavított file pedig *C*. A javítás abból

áll, hogy a javítandó rekordokat a javító file-ról, a többit a javítandó file-ról másoljuk át.



15. ábra

A 15. ábrán az *A* és *B* file-ok szerkezetét ábrázoljuk. A feladat szerint olyan programot kell írnunk, amely a javítandó file összesen 13 rekordját, mégpedig a 2., 7., 12., ... sorszámúakat a javító file-ról vett megfelelő rekordokkal pótolja, míg a többi rekordok a javítandó file-ról másolódnak át.

A megoldás a következő program:

```
        MASTER UPDT
        DIMENSION REK1 (250), REK2 (250)
        READ (5) NREK
        DO 1 I=1, NREK
        READ (5) NO2, REK2
  2     READ (6) NO1, REK1
        IF(NO2-NO1) 3, 1, 3
  3     WRITE(7) NO1, REK1
        GO TO 2
  1     WRITE(7) NO2, REK2
        IF( NO2 .EQ.60) GO TO 4                 (6.32)
        DO 5 I= NO2 , 59
        READ (6) NO1, REK1
```

```
5       WRITE (7) NO1, REK1
4       ENDFILE 7
        REWIND 5
        REWIND 6
        REWIND 7
        STOP
        END
```

Programunk a javító rekordok számának (NREK) beolvasásával kezdődik. Az ezt követő, I szerinti ciklus a beolvasott rekordszámig fut, és végrehajtatja a javítandó file másolását a *C* file-ra. Minden egyes I szerinti ciklusmenet egy újabb javító rekord beolvasásával kezdődik, és ennek az output file-ra történő kiírásával végződik. Amikor a program már valamennyi javító rekordot beolvasta, egy külön programrész segítségével átmásolja a javítandó file utolsó rekordjait, majd lezárja az output file-t, visszacsévéli mindhárom szalagot és megáll.

Ebben a programban erősen kihasználtuk, hogy a javító file-ban a rekordok sorszámai növekvően követik egymást. Ha ez nem teljesül, a feladat jóval bonyolultabbá válik. A program ilyen irányú általánosításával a 2. gyakorlatban foglalkoztunk (lásd 293. old.).

4. Karakterátvitelek. A FORMAT utasítás

Az előző pontban megismerkedtünk a bináris átvitelekkel; ezek jellemzője az, hogy az átvitelben résztvevő perifériális egységen az információ ugyanolyan ábrázolású, mint ahogyan a központi memóriában megjelenik; vagyis az átvitelt nem kísérik konverziós vagy kódváltási műveletek. Ezzel szemben a karakterátvitel mindig együtt jár a külső ábrázolásmódról a belsőre való áttéréssel, ami általában kód- és számrendszerváltást jelent. (Mint látni fogjuk, bizonyos átvitelfajtáknál csak kódváltást.)

Mint már az 1. pontban is jeleztük, a karakterátvitelek minden esetében specifikálni kell az átvitellel együtt elvégzendő kód- és számrendszerkonverziós műveleteket, valamint a re-

kord elrendezését a külső tárolón. Ezt az információt az I/O utasításhoz mellékelt FORMAT utasítás tartalmazza. A FORMAT utasítás karakterátvitelek esetében részben átveszi a rekordhossz meghatározását is. Annyi a karakterátvitelekre is érvényes, hogy az I/O lista kezdete egyben rekordkezdetet és vége egyben rekordvéget is jelent, az „egy I/O lista — egy rekord" elv azonban már nem áll fenn, amennyiben a FORMAT utasítás rendelkezhet úgy is, hogy az I/O listának a külső tárolón egynél több rekord feleljen meg.

A karakterátviteli I/O utasítások formáját a 2. pont (6.1) és (6.3) képleteivel már megadtuk. Az utasításokban szereplő *f* az I/O utasításhoz rendelt FORMAT utasítás címkéje, vagy pedig valamely programbeli tömb azonosítója. Ugyanaz a FORMAT utasítás lehet egyszerre hozzárendelve több I/O utasításhoz is, más szóval, ugyanaz az *f* címke több I/O utasításban is előfordulhat. Mivel a FORMAT utasítások önmagukban sohasem fordulnak elő, hanem mindig csak valamely I/O utasítással kapcsolatban, FORMAT *utasítás sohasem állhat címke nélkül.*

A FORMAT utasítás alakja a következő:

$$f \text{ FORMAT (F)} \qquad (6.33)$$

ahol *f* előjeltelen egész (címke), F pedig az ún. FORMAT-lista. A FORMAT listát mindig kerek zárójelbe zárjuk.

Formálisan az F lista egymástól *elhatároló jelekkel* elválasztott specifikációk vagy specifikációcsoportok sorozata, elején és végén tetszőleges számú törtvonal-karakter is állhat. Elhatároló jel a vessző és a törtvonal. A vessző a FORMAT listában kizárólag elhatároló jelként lép fel, ezzel szemben a törtvonalnak van önálló szemantikai jelentése is (új rekord kezdése — lásd a *j)* alpontot).

A FORMAT listán kétféle specifikáció fordulhat elő:

1. konverziós specifikációk;
2. szerkesztési specifikációk.

A konverziós specifikációkhoz mindig tartozik egy — az I/O lista által meghatározott — adat, és a specifikáció az ezen elvégzendő számrendszer-, vagy kódváltást határozza meg. A konverziós specifikációk és az I/O lista elemei között pontos megfelelkezésnek kell fennállnia, amit a FORTRAN nyelv

szemantikai szabályai részletesen meghatároznak, és amelyekre még visszatérünk.

Ellentétben a konverziós specifikációkkal, a szerkesztési specifikációkhoz nincs az I/O listának egyetlen eleme sem hozzárendelve. Ez utóbbiak célja a külső tárolón a rekord szerkezetének meghatározása: meghatározzák a rekord „külső formáját", azaz pl. a sornyomtató esetében a nyomtatási képet. Az alábbiakban szükségünk lesz még két fogalomra, mégpedig a *mező* és a *mezőszélesség* fogalmára. Mezőn a rekordnak azt a részét értjük, amely a FORMAT lista egyetlen elemének vezérlete alatt kerül átvitelre. Ha ez az elem konverziós specifikáció, akkor a megfelelő mező a rekordon egy számadat. Az ilyen mezőt *numerikus mezőnek* nevezzük. A sornyomtatón pl. az egy sorban kinyomtatott számok mindegyike egy-egy numerikus mező. Ha a specifikáció szövegátvitelt ír elő, akkor ennek a rekordon *alfanumerikus* (vagy *Hollerith*) *mező* felel meg. A sornyomtató példájánál maradva, az eredménytáblázat fejléceiben szereplő szövegek, feliratok Hollerith (alfanumerikus) mezők.

Valamely mező szélességén a mező által elfoglalt karakterpozíciók számát értjük. Numerikus mezők szélességének kiszámításakor a numerikus mezőhöz tartozó minden karaktert (előjel, tizedespont stb.) figyelembe kell venni. A továbbiakban a mezőszélességet mindig w-vel jelöljük. Például a $-5.3047E-02$ számra $w=11$.

Bevezetjük még a következő jelölést is: bármely szám tizedespont utáni tizedesjegyeinek számát d-vel jelöljük*. d-be csak a decimális mantissza törtrészének jegyeit számítjuk bele, az esetleges decimális exponens jegyeit nem. Így pl. az előbbi számpéldában $d=4$.

A soron következő alpontokban rendre áttekintjük az egyes specifikációtípusokat. Az 5. pontban visszatérünk a FORMAT lista egészét érintő szabályok ismertetésére, a 6. pontban pedig, amikorra már a FORTRAN nyelvet áttekintettük, két teljes mintaprogramot mutatunk be.

* A *d)* alpontban *d*-nek, *g)*-ben pedig *w*-nek a megállapodástól eltérő jelentést tulajdonítunk!

a) Egész számok konverziója. Az I specifikáció

Egész (INTEGER) típusú változók értékeinek beolvastatásához és kiíratásához az I konverziós specifikációt alkalmazzuk, melynek alakja:

$$\mathrm{I}w \qquad (6.34)$$

ahol *w* — megállapodásunknak megfelelően — a mezőszélességet jelenti, vagyis azt határozza meg, hogy a perifériális rekordokon az átviendő egész szám hány karakterpozíciót foglal el.

Output I specifikációval

Kiírási típusú átvitelek során, ha az átvinni kívánt szám nem tartalmaz annyi számjegyet, amennyi a *w* mezőszélességnek megfelel, a gép a számot a mező jobb oldali, azaz magasabb sorszámú karakterpozícióira írja ki, a bal oldali, fel nem használt karakterpozíciókra pedig betűköz karakterek (space) kerülnek*.

Példa

Legyenek a programnak K1, K2 és K3 egész típusú változói, aktuális értékeik pedig rendre legyenek 12397, −3540 és 52 888 Írjunk WRITE utasítást, amely a sornyomtatón kinyomtattatja ezeket az értékeket! Feltesszük, hogy a sornyomtató logikai perifériaszáma 2.

Megoldás. A későbbiekben sokat foglalkozunk majd az output formátum megtervezésével, néhány probléma azonban már e legelső példa kapcsán is felmerül.

Mielőtt megtervezzük a nyomtatási képet, vagyis az output formátumot, mindenekelőtt meg kell becsülnünk a kiíratandó egész mennyiségek nagyságrendjét, mert e nélkül nem tudjuk megállapitani a mezőszélességeket. Általános elvként azt tartsuk szem előtt, hogy egy $w-1$ jegyű egész szám kiírására az output rekordon legalább *w* karakterpozíciót kell fenntartanunk, egy karakterpozíciót ui. az előjel számára biztosítunk.

Tegyük fel, hogy K1, K2 és K3 abszolút értéke maximálisan 99999 lehet, azaz e számok legfeljebb ötjegyűek. (Történetesen

* A betűköz karaktereket a továbbiakban, ahol szükséges, sp-vel (space) rövidítjük.

pillanatnyilag K2 aktuális értéke négyjegyű, de erre természetesen nem lehetünk tekintettel.) Előző szabályunk szerint legalább 6 karakterpozíciót kell az output rekordon mindegyikük számára fenntartanunk. Nézzük meg, milyen nyomtatási képhez jutunk, ha $w=6$-ot választunk! Legyen a kiírási utasítás, a megfelelő FORMAT utasítással együtt,

```
      WRITE (2, 15) K1, K2, K3
15    FORMAT (I6, I6, I6)
```
(6.35)

(6.35) végrehajtása után a következő nyomtatási képhez jutunk (itt és a következőkben a nyomtatási képen a papír bal szélét egy függőleges vonal szimbolizálja):

|▽12397▽ −3540▽52888 (6.36)

E nyomtatási képpel kapcsolatban megjegyezzük először is, hogy a pozitív előjel helyett mindig betűköz karakter nyomtatódik. Így az első számot a lapszéltől, valamint a harmadik számot a másodiktól elválasztó egyetlen betűköz a pozitív előjel helyett áll. Az első számot a másodiktól elválasztó betűköz ezzel szemben azért került az output rekordra, mert K2 nem tölti ki a rendelkezésre álló hat karakterpozíciót, hanem csak ötöt, lévén négyjegyű szám negatív előjellel. Emlékeztetünk iménti szabályunkra, mely szerint ilyen esetben a mező bal szélén keletkeznek a betűköz karakterek. Ezt úgy fejezzük ki, hogy *kiírási átvitelek során a numerikus mezők az output rekordon jobbról tömörítve jelennek meg.*

Példánkra visszatérve, látjuk, hogy a kapott nyomtatási kép nem nagyon tetszetős, mivel a számok szinte egymásra vannak zsúfolva. Különösen szembetűnne ez, ha történetesen K2 értéke nem −3540 volna, hanem −35400. Ebben az esetben ui. a (6.35) kiírási utasításpár hatására a következő nyomtatási kép jönne létre:

|▽12397 −35400▽52888 (6.37)

azaz K1 és K2 között egyetlen elválasztó karakter sem állna.

Tetszetősebb a nyomtatási kép, ha kihasználjuk a jobbról tömörítési szabályt az output rekord „levegősebbé” tételére. Ezt azzal érhetjük el, hogy az I specifikációban a minimális-

nál nagyobb *w* értéket adunk meg. Ha pl. (6.35)-ben a FORMAT utasítást ekként módosítjuk:

```
15    | |FORMAT(I11, I11, I11)                (6.38)
```

akkor kiírási képünk a következő lesz:

|▽▽▽▽▽▽12397▽▽▽▽▽▽ –3540▽▽▽▽▽▽52888

(6.39)

Az egyes számok most már jól elkülönülnek, mert legalább öt betűköz karakter választja el őket.

A jobbról tömörítési szabályt felhasználhatjuk a kiírt számok középre állítására is, ha ismerjük a sornyomtató egy sorában a karakterpozíciók számát. Az általánosan használatban levő sornyomtatókon ez a szám 120 vagy 160. A továbbiakban feltesszük, hogy a rendelkezésünkre álló sornyomtató 120 pozíciós. Módosítsuk feladatunkat úgy, hogy írjuk elő a három szám sorközépre állítását is!

Ha fenntartjuk azt a kikötésünket is, hogy az egyes számokat 5—5 betűköz karakter válassza el, ha elfoglalják a maximális hat karakterpozíciót, akkor a következőképpen kell számolnunk: a K1-nek megfelelő numerikus mező hat pozíciót foglal el az output rekordon, K2 és K3 pedig egyenként 11—11 pozíciót, a három numerikus mező együttesen tehát összesen 28-at. Fennmarad tehát $120-28 = 92$ pozíció, ezt kell megfeleznünk a bal és a jobb margó között. Végeredményben a három számot a következő FORMAT utasítással állíttathatjuk sorközépre:

```
15    | |FORMAT(I52, I11, I11)                (6.40)
```

A fenti formátum hatására 46 betűköz karakter nyomtatódik K1 elé, ami által a három szám sorközépre kerül.

Annak érdekében, hogy belássuk, hogy (6.40) valóban úgy működik, ahogy várjuk tőle, hivatkozunk a 4. pont bevezetőjében említett szemantikai szabályra, amely szerint az I/O lista kezdete rekordkezdetet, vége pedig rekordvéget jelent. Amikor tehát a gép elkezdi végrehajtani a (6.35)-ben szereplő WRITE utasítást, mindenekelőtt új rekordot nyit, ami a sornyomtató esetében új sor kezdését jelenti. Ezt követőleg átviszi (azaz kinyomtatja) az I/O lista első elemét, a FORMAT utasítás első

konverziós specifikációjának megfelelően. Ez esetünkben az I52 specifikáció; az output rekord első, 52 karakter szélességű numerikus mezejének utolsó hat karakterében elhelyeződik a K1 szám aktuális értéke (az előjel helyett itt szintén betűköz karakter áll). A második I/O listaelem a második konverziós specifikáció szerint nyomtatódik, ez a K2 szám, a neki megfelelő specifikáció I11. Az eljárás hasonló az előzőhöz, a kapott numerikus mező első öt karakterpozíciójára betűköz kerül, a többi a kinyomtatandó számot tartalmazza (itt a negatív előjel is elfoglal egy pozíciót). Ugyanígy történik a harmadik szám kinyomtatása is. Több I/O listaelem nem lévén, a sornyomtató e sorának további részeibe nem kerül információ, így automatikusan létrejön az ugyancsak 46 karakter szélességű jobb margó.

Amint az előbbiekből látható, a karakterátvitelek végrehajtása az I/O és a FORMAT lista párhuzamos átvizsgálása és értelmezése alapján történik. Ennek a műveletnek elég bonyolult „játékszabályai" vannak, melyeket az 5. pontban tárgyalunk.

A FORMAT utasítások írásának kényelmesebbé tétele céljából bevezették az egyes specifikációk elé írható ún. *ismétlési tényezőket*. Az ismétlési tényező egy előjeltelen egész szám, amely azt jelenti, hogy az adott specifikációt az átvitel folyamán hányszor kell figyelembe venni. A (6.38) specifikációt például a következőképpen rövidíthetjük:

```
15    | |FORMAT(3I11)                                    (6.41)
```

a (6.40) specifikáció lehetséges rövidítése pedig:

```
15    | |FORMAT (I52, 2I11)                              (6.42)
```

Ismétlési tényezőt tehát akkor alkalmazhatunk, ha két vagy több teljesen megegyező specifikáció kerül egymás mellé a FORMAT listán. Mint az előző példán is láthattuk, az *n*S alakú listaelem — ahol az *n* előjeltelen egész szám az ismétlési tényező, S pedig tetszőleges specifikáció H (lásd *h)*) és / (lásd *j)*) kivételével — definíciószerűen egyenértékű az

$$\underbrace{S, S, \ldots, S}_{n\text{-szer}} \qquad (6.43)$$

listarészlettel.

Input I specifikációval

Vizsgáljuk most az I specifikáció jelentését *beolvasási* műveletek esetére. Általában, ha egy konverziós specifikáció hatását beolvasási műveletekre tárgyaljuk, mindig két kérdést kell tisztáznunk:

1. az adott specifikációhoz milyen számok engedhetők meg az input információhordozón;

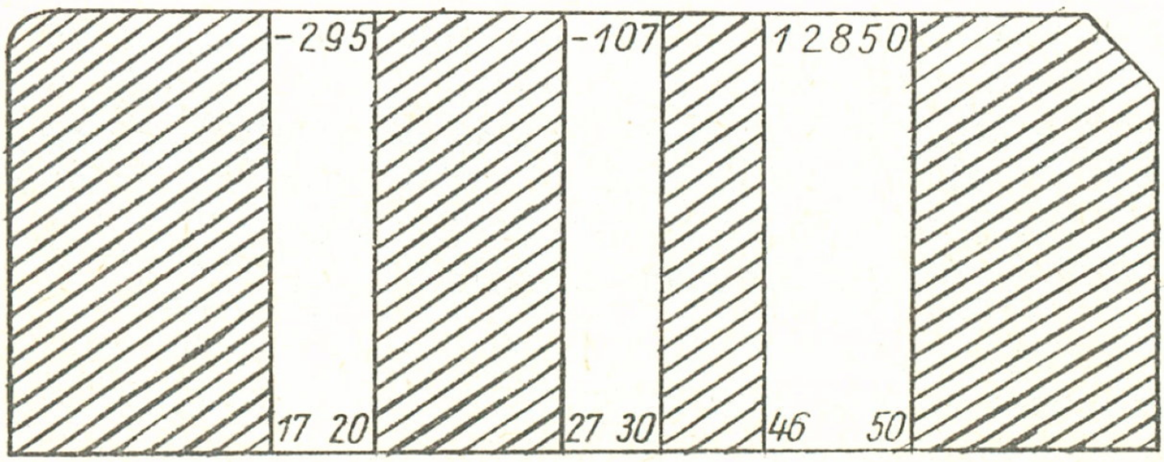
2. az adatokat a beolvasáskor a gép hogyan értelmezi.

Ami az első kérdést illeti, az I specifikáció esetében *az input periférián megjelenő numerikus mezőnek a* FORTRAN *nyelv definícióinak megfelelő egész számnak kell lennie* (vagy előjellel ellátva, vagy anélkül).

A beolvasott információ értelmezésének tekintetében már bizonyos óvatossággal kell eljárnunk. A kiviteli művelet tárgyalása során megismertük a jobbról tömörítés szabályát. Lényegében ez érvényes a beolvasásra is: *az input rekordon megjelenő numerikus mezőt a gép jobbról tömörítettnek tekinti;* a mező bal szélén álló esetleges betűköz karaktereket a beolvasáskor nem veszi figyelembe, a jobb szélen állókat pedig zérusnak tekinti.

Példák

1. Álljon az input kártya 17—20. oszlopában a −295, a 27—30. oszlopában a −107, a 46—50. oszlopában pedig a 12850 szám. (Lásd a 16. ábrát!) Írjunk FORTRAN utasítást e három szám beolvastatására!



16. ábra

Megoldás:

```
   |  |READ (1,20) K1, K2, K3
20 |  |FORMAT (I20, I10, I20)
```
(6.44)

Feltettük, hogy a kártyaolvasó logikai perifériaszáma 1, és a beolvastatandó mennyiségek K1, K2 és K3. A jobbról tömörítési szabálynak megfelelően, a specifikációkat úgy választottuk meg, hogy az utolsó számjegy a mező utolsó karakterpozíciójára kerüljön.

2. A tömörítési szabály fontosságának illusztrálása céljából térjünk most vissza a 216. oldalon található példa számadataira. Tegyük fel, hogy ezek a számadatok úgy vannak elhelyezve egy input kártyán, ahogyan azt (6.39) mutatja. Ha ezt a kártyát a következő READ utasítással olvastatjuk be:

```
 |  |READ (1,15) K1, K2, K3
```
(6.45)

ahol a 15-ös FORMAT utasításon (6.38)-at értjük, akkor a gépbe ugyanezek a számok kerülnek be. Ha ezzel szemben (6.45)-höz a következő FORMAT utasítást rendeljük:

```
15 |  |FORMAT (3I12)
```
(6.46)

akkor ennek hatására K1 értéke 123970 lesz, K2 értéke—354000 K3 értéke pedig 52888000, ebben az esetben ugyanis a szám utolsó számjegye nem esik egybe a mező utolsó karakterpozíciójával, az ott álló betűköz karakterek zérusként értelmeződnek, és a beolvasandó számnak 10-zel, 100-zal, ill. 1000-rel való szorzását eredményezik.

Végezetül kimondunk még egy általános elvet, amely a FORMAT utasítás szemantikájában következetesen végigvonul: *ha egy karakterátviteli* WRITE *utasítással kiíratunk bizonyos mennyiségeket, majd ugyanezeket a mennyiségeket azonos* FORMAT *szerint visszaolvastatjuk, akkor — az esetleges kerekítési hibáktól eltekintve — ugyanazokat a számokat kell visszakapnunk.* Nincs tehát semmi meglepő abban, hogy a (6.45) READ utasítás a (6.38) FORMAT-tal együtt a (6.35) alatti WRITE utasítással kiíratott számokat változatlanul viszi vissza. A (6.46)-tal kapcsolatban mondottak viszont arra hívják fel a figyelmet, hogy ha a WRITE-hoz és a READ-hez

nem ugyanaz a FORMAT tartozik, akkor a visszavitel eredménye különbözhet a kiírás előtti értékektől.

Ez az elv magyaráz meg egyes szemantikai szabályokat, amelyek önmagukban esetleg első pillanatban illogikusnak tűnhetnek (pl. a P kiegészítő specifikációra vonatkozó szabályok — lásd a Hatodik fejezet 12. pontját).

b) Fixpontos, valós számkonverzió. Az F specifikáció

Az előző alpontban megismert I konverziós specifikáció akkor alkalmazható, ha a megfelelő I/O listaelem egész típusú változó. Valós változók átvitelére háromféle konverziós lehetőségünk van, mégpedig az F specifikáció, melyet a jelen alpontban tárgyalunk, továbbá az E specifikáció (*c)* pont) és a G specifikáció (*d)* pont).

Az F specifikáció alakja a korábban bevezetett jelölésekkel:

$$\mathrm{F}w\cdot d \tag{6.47}$$

Output F specifikációval

Kiírási utasításokban a *d* szám azt mondja meg, hogy hány tizedesjegyre kell a kiíratandó változó aktuális értékét kerekíteni. A *w* mezőszélességre ugyanaz érvényes, mint az I specifikáció esetében: a gép a kiíratandó számot a rendelkezésre álló output mezőben *jobbról tömörítve* helyezi el, és a mező bal oldalán megmaradó, fel nem használt karakterpozíciók betűköz karakterekkel töltődnek fel.

Példák

1. Legyenek valamely programban X, Y és Z valós típusú változók; tegyük fel, hogy $|X|<10$ és 8 tizedesjegyre kell kiíratni, $|Y|<10^5$ és 2 tizedesjegyre kell kiíratni, végül $|Z|<100$ és 4 tizedesjegyre kell kiíratni. Írjunk megfelelő WRITE utasítást!

Megoldás. Az output formátumot az I specifikáció kapcsán ismertetett elvek szerint kell megterveznünk, figyelembe véve az egyes kiíratandó mennyiségekre vonatkozó korlátokat. Ezek alapján most is megállapítható a *w* mezőszélesség minimális

értéke, de ehhez a tizedesjegyek előírt számát is figyelembe kell vennünk.

Az X mennyiség kiírásakor egy egész és 8 tizedesjegyre kell számítanunk, ezenkívül egy karakterpozíciót elfoglal maga a tizedespont, egyet pedig az előjel. Összesen tehát legalább 11 karakterpozícióra van szükség; $w \geqq 11$. Y esetében az egész rész 4 karakterpozíciót foglalhat el, a törtrész kettőt, a tizedespont és az előjel ismét 1—1 pozíciót, vagyis Y-ra $w \geqq 8$. Hasonló meggondolás alapján Z-re ugyancsak $w \geqq 8$. Általában fennáll, hogy ha valamely Q számot, melynek abszolút értéke 10^n-nél határozottan kisebb, k számú tizedesjegyre kívánunk kiíratni, akkor $w \geqq n+k+2$. Ezt a szabályt azonban még kiegészítjük azzal, hogy valódi tizedestörtek kiírásakor, vagyis $n=0$ esetében, kívánatos a minimális $w = k+2$ helyett legalább $w = k+3$ választása. Ha ugyanis elegendő hely van, akkor a valódi tizedestörteket a gép zérus egész résszel írja ki, vagyis az egész rész helyén elhelyez egy nullát. Ha erre nem biztosítunk helyet, a tizedespont közvetlenül az előjel után következik. Ezt jobb elkerülni, mert így a nyomtatási kép bizonyos esetekben nem lesz egyöntetű.

Feladatunkra visszatérve, ennek „minimális" megoldása:

```
        WRITE (2,5) X, Y, Z
5       FORMAT (F11.8, F8.2, F8.4)
```
(6.48)

Tegyük fel, hogy (12 értékes decimális jegyre történő belső számábrázolást feltételezve) X aktuális értéke −0.363844055713, Y aktuális értéke 3892.18485004 és Z aktuális értéke −20.0996337781. Ekkor (6.48) hatására a következő nyomtatási kép jön létre:

```
|−0.36384406▽3892.18−20.0996
```
(6.49)

A (6.40) formátumhoz hasonlóan, ismét előírhatjuk a kiírt számanyag középre állítását (120 pozíciós sornyomtatót tételezünk fel). Kössük ki, hogy az egyes számadatokat legalább 5 betűköz karakternek kell elválasztania; ekkor a (6.48)-ban szereplő FORMAT-ot ekként módosíthatjuk:

```
5       FORMAT (F53.8, F13.2, F13.4)
```
(6.50)

Eljárásunk hasonló a (6.40)-nel kapcsolatban követetthez: a második és a harmadik specifikációban *w* értékét megnöveltük 5-tel, ami biztosítja, hogy a két szám közé legalább 5 betűköz karakter kerüljön (jobbról tömörítés!); ezután összeadtuk az egyes változók számára szükséges *w* értéket: $11+13+13 = 37$; ezt a számot levontuk 120-ból, hogy megkapjuk a kihasználatlan karakterpozíciók számát: $120-37 = 83$; ezt „megfelezve" kapjuk a bal margó szélességét ($=42$); ezzel megnövelve az első specifikációban *w*-t, éppen a (6.50) formátumhoz jutunk.

A (6.50) formátum alapján a következő nyomtatási kép alakul ki:

$$|\underbrace{\triangledown\triangledown\ldots\triangledown\triangledown}_{42\text{ sp}}-0.36384406\triangledown\triangledown\triangledown\triangledown\triangledown\triangledown3892.18\triangledown\triangledown\triangledown\triangledown\triangledown-20.0996$$

2. Eddigi ismereteink birtokában most hozzáfoghatunk egy jóval bonyolultabb kiíratási feladathoz. Tegyük fel, hogy adva van a memóriában egy 12×12-tes mátrix, és ezt kell sorfolytonosan kiíratnunk. A sornyomtató egy sorában a mátrix egy sora kell hogy álljon, és a sorokat meg is akarjuk számozni, mert ezzel a nyomtatási kép áttekinthetőbbé válik. Feltesszük, hogy a mátrix elemei abszolút értékben kisebbek, mint 100; a kiírást 3 tizedesjegyre követeljük meg.

Megoldás. Mindenekelőtt kiszámítjuk, hogy egy-egy sorban hány karakterpozíciót kell felhasználni. Minthogy egy sorban 12 valós szám és egy (egész) sorszám nyomtatását követeljük meg, és az előzők szerint a nyomtatáshoz minden egyes valós mátrixelem számára legalább $2+3+2 = 7$ pozícióra van szükségünk, a sorszámnak pedig legalább két* pozíciót kell fenntartanunk, tehát soronként legalább $12\times7+2 = 86$ nyomtatási pozícióra van szükségünk; így elválasztó betűközökre és margóra marad $120-86 = 34$ karakterpozíció. 13 darab egy sorban nyomtatott számnak 12 „köze" van; ha számonként 2—2 elválasztó betűközzel számolunk, akkor margóra marad 10 pozíció, amit megfelezve, 5—5 karakter szélességű bal és jobb margóhoz jutunk.

* A sorszám mindig pozitív, így az előjel számára jelen esetben nem szükséges helyet kihagynunk.

Mindezen meggondolások figyelembevételével, a kiírási programrészlet a következő:

```
      DO 12 I = 1, 12
12    WRITE (5,8) I,(A(I,J),J = 1,12)
8     FORMAT (I7, 12F9.3)
```
(6.51)

A kiírást a 12-es címkéjű WRITE utasítás realizálja. Az utasítás I/O listája egy egyszerű és egy összetett listaelemből áll. Az I/O listán szereplő I ciklusváltozó kinyomtatása útján kapjuk a lap bal szélén a sorszámot.

Annak megértéséhez, hogy miért szerveztük a ciklust éppen így, azaz miért helyeztük a sor (I) szerinti ciklust egy DO utasítás vezérlete alá, és miért van az oszlop (J) szerinti ciklus az I/O listában elhelyezve, emlékeztetünk arra a szemantikai szabályra, amely szerint az I/O utasítás végrehajtásának megkezdése új rekord megnyitásával jár együtt (lásd a 4. pont bevezetését, 214. old.). A (6.51) szervezési megoldás tehát automatikusan biztosítja, hogy a gép a mátrix minden egyes sorának kinyomtatását új sornyomtató sorban kezdje, a mátrix azonos sorában álló elemek pedig azonos sornyomtató sorba kerüljenek.

A későbbiekben látni fogjuk, hogy az explicit DO ciklusra valójában nincs szükség, korrekt megoldás a következő is:

```
      WRITE (5,8) (I, (A(I, J),J = 1,12),  I = 1,12)
8     FORMAT (I7, 12F9.3)
```
(6.52)

Ezen utasításpár helyes működését az 5. pontban ismertetendő szabályok biztosítják.

Input F specifikációval

Mindenekelőtt emlékeztetünk arra, hogy beviteli utasítások végrehajtásakor a gép a beolvasott numerikus mező bal oldalán álló betűköz karaktereket figyelmen kívül hagyja, míg a mező jobb oldalán álló betűköz karaktereket zérusnak tekinti (lásd a (6.45) és (6.46) utasításokkal kapcsolatban mondottakat). Ez a szabály érvényes a valós fixpontos specifikációra is, tehát ha a beolvasott numerikus mezőben álló szám nem jobbról

tömörített, akkor beolvasáskor az alacsony helyi értékű pozíciókban megfelelő számú zérussal egészül ki.

Mielőtt ennek következményeit kifejtenők, mindjárt kimondjuk a valós beolvasási konverzió két további alapszabályát is:

1. *Ha az* F *specifikáció mellett beolvasott numerikus mezőben van tizedespont-karakter, akkor a specifikációban szereplő d mennyiség* (amely kiviteli műveletekben a törtrész tizedesjegyeinek számát határozza meg) *a beolvasás szempontjából közömbös, és a konverzió végrehajtását nem befolyásolja;*

2. *Ha a beolvasott mezőben nincs tizedespont-karakter* (azaz a beolvasott szám formailag egész), *akkor d az „elvágandó" tizedesjegyek számát határozza meg.*

Az F konverzióval való bevitelkor az input mező tetszőleges olyan számot tartalmazhat, amely az ASA FORTRAN nyelv definíciói értelmében „szám"-nak (azaz egész, vagy valós számnak) tekinthető. Tehát nem követeljük meg, hogy az input mező pontosan ugyanolyan legyen, mint amilyent ugyanaz az F specifikáció egy kiviteli utasításban létrehozott volna.

E szabályoknak a valós számok bevitelére nézve két fontos következménye van:

1. ha a beolvasott valós szám tartalmaz tizedespontot, akkor nem feltétlenül kell az input mezőben jobbról tömörítve elhelyezkednie, mert a jobb oldali kiegészítő zérusok „értéktelenek", és a számot nem változtatják meg;

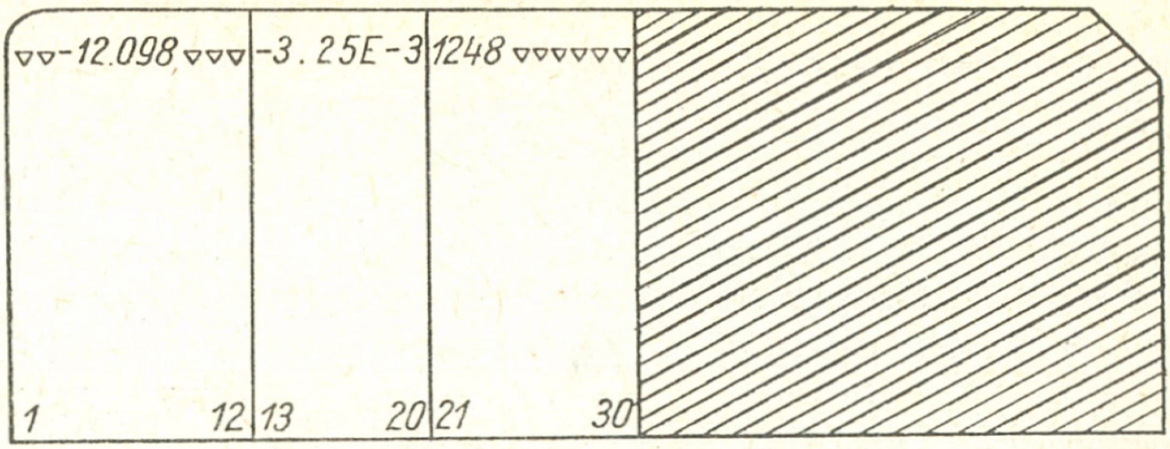
2. ha viszont a beolvasott szám nem tartalmaz tizedespontot, akkor a jobbról tömörítés lényeges, mert a kiegészítő zérusok — akárcsak az egész számok esetében — a beolvasott számnak 10 megfelelő hatványával való szorzását jelentik.

Példák

1. Legyenek X, Y és Z valós változók, és tegyük fel, hogy ezekre a következő input utasítást adjuk ki:

```
  |  |READ (4,3) X, Y, Z
3 |  |FORMAT (F12.4, F8.2, F10.3)
```
(6.54)

Feltesszük, hogy a 4-es logikai perifériaszámú berendezés a lyukkártyaolvasó, és a soron következő input kártya a 17. ábrán



17. ábra

látható alakú. Kérdés, mi lesz a (6.54) utasítás végrehajtásának befejezése után X, Y és Z értéke?

Megoldás. A lyukkártyán található első — 12 karakter szélességű — numerikus mezőben, nem jobbról tömörített formában, a -12.098 szám áll. Az ismert szemantikai szabályok értelmében az elöl álló betűközöket beolvasáskor nem vesszük figyelembe, a hátul állókat viszont zérusként értelmezzük. A szám tehát valójában nem -12.098, hanem -12.098000. Az első F specifikációban található *d* mennyiség (melynek értéke példánkban 4) nem befolyásolja a bevitelt, mert az input mezőben van tizedespont, ezért a tizedesjegyek számát nem a specifikáció határozza meg.

A következő szám a lyukkártyán teljes egészében kitölti a rendelkezésére álló 8 karakter szélességű mezőt. A szám $-3.25E-3$, ez is tartalmaz tizedespontot, így a második specifikációban sincs jelentősége annak, hogy *d* éppen 2.

Az input kártya harmadik adata az 1248 egész szám, utána 6 betűköz karakterrel. A szám után álló betűköz karakterek ismét zérusokat jelentenek, tehát a szám valójában 1248000000. Ebben az esetben a specifikációból tudjuk meg, hogy hány tizedesjegyet kell levágni: hármat, így a beolvasott szám 1248000.

Összegezve, a beolvasás utáni aktuális értékek rendre: $X = -12.098$, $Y = -0.00325$ és $Z = 1\,248\,000$. Természetesen ezek a számok a belső memóriában az illető gépnek megfelelő — általában lebegőpontos bináris — számábrázolásban tárolódnak.

2. Legyenek egy 12×12-es tömb elemei lyukkártyára lyu-

kasztva. Egy-egy lyukkártyán 12 szám van, és ezek a mátrixnak egy sorában vannak. Összesen tehát 12 lyukkártyán vannak a mátrixelemek sorfolytonosan rögzítve. Az egyes számok 6 karakter szélességű mezőkben vannak elhelyezve, jobbról tömörített, *tizedespont nélküli* (egész) ábrázolásban. A lyukkártyák utolsó 8 karakterpozíciója nincs kihasználva. A számok egységesen 4 tizedesjegyre vannak megadva (vagyis ennyit kell levágni). A feladat olyan FORTRAN programrészlet írása, amely beolvastatja ezt a tömböt.

Megoldás. Tekintve, hogy a számok nem tartalmaznak tizedespontot, az input formátumban a mezőszélességen kívül a levágandó tizedesjegyek számát is specifikálni kell: $d=4$. A megfelelő beolvasási specifikáció tehát F6.4. A beolvasási program:

```
      DO 2 I=1,12
2     READ (4,9)(A(I,J), J=1,12)                (6.55)
9     FORMAT (12F6.4)
```

A program működésével kapcsolatban utalunk a (6.51) programnál mondottakra: az I szerinti DO ciklus biztosítja minden egyes mátrixsor beolvasásánál új rekord (adott esetben: új kártya) kezdését.

Valószínűleg mind ebben, mind a korábbi példákban feltűnt az Olvasónak, hogy az I/O utasításokban nem intézkedünk az egyes rekordok végén kihasználatlanul, vagy figyelmen kívül hagyott karakterpozíciók sorsáról (mint amilyenek pl. a kiírási példákban a jobb margó, vagy a jelen példában az egyes kártyák jobb szélén álló 8 karakterpozíció). Ez így rendjén is van, mert *a* FORMAT *utasítás specifikálhat kevesebb karakterpozíciót, mint amennyit a rekord ténylegesen tartalmaz.* Kiírási utasításokban az ily módon definiálatlanul maradó jobb oldali karakterpozíciók betűköz karakterekkel töltődnek fel, beolvasási utasításokban pedig figyelmen kívül maradnak. A (6.51)-ban szereplő 8-as címkéjű FORMAT pl. csak 115 karakterpozíciót specifikál, így keletkezik az 5 karakter szélességű jobb margó; (6.55)-ben pedig a FORMAT rendelkezései 72 karakterpozícióra szólnak, ami egyértelmű az utolsó nyolc karakterpozíció ignorálásával (a lyukkártyát 80 oszloposnak tételezzük fel).

E szabályokkal kapcsolatban két megjegyzést kell még tennünk:

1. Olyan perifériatípusokra, amelyek rekordhossza nem rögzített (mint pl. a lyukszalag vagy a mágnesszalag), a szabálynak csak a beolvasásra vonatkozó része érvényes; az ilyen perifériákon kiíráskor mindig annyi karakterpozícióból álló rekord keletkezik, amennyit a FORMAT utasítás ténylegesen specifikál.

2. Az adatelőkészítés (lyukasztás) során gondosan ügyelni kell arra, hogy az input rekordok egyike se legyen rövidebb, mint amennyit a FORMAT utasítás előír, mert ha az input rekord beolvasáskor a kelleténél rövidebbnek bizonyul, ez hibát jelent, amely után a gép megszakítja a futtatást.

Ha egy beolvasásra kerülő valós mennyiség esetében az input mező decimális exponenst is tartalmaz, akkor a kitevőben a pozitív előjel betűközzel helyettesíthető. (Tehát nemcsak a szám pozitív előjele helyett állhat betűköz, hanem a kitevő pozitív előjele helyett is.)

Az ASA FORTRAN egyébként úgy rendelkezik, hogy numerikus mezőben minden betűköz karakter zérusnak számít, kivéve a szám elején állókat, amelyek ignorálandók. A legtöbb gépi reprezentáció azonban a számok belsejében álló betűközöket illegális karakterként jelzi, ezért zérusoknak betűközzel való helyettesítését nyomatékosan ellenjavalljuk.

Komplex számok átvitelére nincs külön konverziós specifikáció. Komplex számok bevitelét és kiíratását valós konverziós specifikációk segítségével vezérelhetjük; az I/O lista minden komplex eleméhez két valós konverziós specifikációnak kell tartoznia, egynek a valós, és egynek a képzetes rész konverziójára. A komplex számokat tehát úgy visszük át, mint két egymástól független valós számot.

Ebben az alpontban a valós beviteli konverzióra vonatkozó tudnivalókat az F specifikációra mondottuk el. Ezek azonban egyben az E és a G specifikációkra is vonatkoznak (lásd a következő két alpontot), mivel *beviteli utasítások esetén az* F, *az* E *és a* G *specifikáció teljesen egyenértékű*, csak a kiírási műveletekre különböző a hatásuk. Ennélfogva a következő két alpontban csak kiírási műveletekkel foglalkozunk.

c) Lebegőpontos, valós számkonverzió, Az E specifikáció

A korábban bevezetett jelölésekkel az E specifikáció alakja

$$\mathrm{E}w.d \tag{6.56}$$

Output E specifikációval

E specifikáció segítségével a program valós változóinak aktuális értékét decimális normálalakban írhatjuk ki, mely egy 0.1 és 1 közé eső, *d* számú tizedesjegyből álló decimális mantisszából, és egy két jegyből álló decimális karakterisztikából tevődik össze. A (6.56) specifikáció hatására kiírt szám alakja tehát

$$\pm 0.\underbrace{yy \ldots y}_{d \text{ számú jegy}}\mathrm{E} \pm yy \tag{6.57}$$

ahol minden *y* egy-egy decimális számjegyet szimbolizál, a + jel helyett pedig egyes gépi reprezentációkban betűköz karakter áll.

(6.57)-ből látható, hogy a decimális exponens összesen 4 karakterpozíciót foglal el az output mezőben, a szám előjele, a kötelező zérus és a tizedespont együttesen hármat, így ha *d* számú jegyet kívánunk kiíratni, akkor legalább $w = d + 7$ kell hogy legyen.

Az E specifikáció előnye az F specifikációval szemben, hogy a szám nagyságától függetlenül mindig előre ismert számú karakterpozíciót igényel. Ha az F specifikáció esetében a kiíratandó szám véletlenül nem olyan nagyságrendű, mint amilyennek vártuk — azaz túl nagy, vagy túl kicsi —, akkor vagy egyáltalán nem fér bele a számára fenntartott mezőbe („kicsordul" belőle), vagy a konverzió közben minden értékes számjegye eltűnik, és zérusként jelenik meg (ez az ún. „alulcsordulás" esete). Az E specifikáció esetében ez nem történhet meg; a számot az output rekordban mindig adott relatív pontossággal (adott számú értékes jegyre) kapjuk.

Példák

1. Legyen adva három, egyenként 20 elemű vektorunk, **A**, **B** és **C**. A vektorok elemeinek nagyságrendje előre nem

ismert. A feladat a vektorok elemeinek kiíratása a sornyomtatón, mégpedig oly módon, hogy mindegyik sorban, középre állítva, három szám álljon, A-nak, B-nek és C-nek egy-egy azonos indexű eleme. Az egyes vektorelemeket 6 tizedesjegyre nyomtassuk. A sorokat számozzuk is meg!

Megoldás. Tekintve, hogy példánkban $d=6$, az egyes valós számok kinyomtatásához legalább $w=13$ mezőszélességet kell előírnunk. A sorszám kétjegyű, tehát kinyomtatása 2 karakterpozíciót igényel. A példa szövege nem írja elő az elválasztó betűköz karakterek számát; a tapasztalat szerint az elválasztó betűközök optimális száma 5—10, mert ilyen értékek mellett a számok már jól elkülönítettek, de a nyomtatási kép még nem folyik szét. Válasszuk önkényesen az elválasztó betűközök számát 8-nak! A már ismert számítási eljárás szerint — az elválasztó betűközöket is beleszámítva — a kinyomtatás 65 karakterpozíciót igényel, margóra tehát 55 pozíció marad. A bal margót 27 karakterpozíció szélességűnek választva, a megoldás a következő:

```
      DO 4 I=1, 20
 4    WRITE (5,12) I, A(I), B(I), C(I)
 12   FORMAT (I29, 3E21.6)
```
(6.58)

2. Írjunk FORTRAN programot az

$$f(x) = \frac{10x}{1-x^2} \tag{6.59}$$

függvény tabellázására, $x=0$-tól $x=0.999$-ig, ezredenként, és nyomtassuk ki az abszcissza- és ordinátaértékeket, utóbbiakat 6 tizedesjegy pontossággal!

Megoldás. Tekintettel arra, hogy a (6.59) függvény az $x=1$ hely közelében igen rohamosan növekszik (magán az $x=1$ helyen végtelenné válik), a függvényérték kinyomtatására célszerű a lebegőpontos formátum alkalmazása. Az abszcissza nyomtatására viszont célszerűbb a fixpontos formátum. A feladat egy lehetséges megoldása (a formátum kiszámításának rész-

letezésétől ezúttal eltekintünk):

```
      MASTER TABL
      DO 2 I=1,1000
      X = 0.001 * FLOAT(I-1)
      F = 10. * X/(1.-X * * 2)
2     WRITE (5,3) X, F                    (6.60)
      STOP
3     FORMAT (F54.3, E18.6)
      END
```

Megjegyezzük, hogy a FORMAT utasítás akárhol állhat a programban, nem kell, hogy közvetlenül a hozzá tartozó I/O utasítás után következzék. Sőt, az áttekinthetőség érdekében sokszor célszerű a FORMAT utasításokat a szegmenst lezáró END elé összegyűjteni.

Szükségesnek tartjuk megjegyezni azt is, hogy a FORMAT — nem végrehajtható utasítás lévén — nem lehet egy DO ciklus lezáró utasítása. A (6.52), (6.55) példákban, valamint a jelen példában is, a DO lezárására mindig végrehajtható utasítást alkalmaztunk. Tilos természetesen a vezérlésátadás is a FORMAT utasításra.

Input E specifikációval

Input utasítások esetén az E specifikáció teljesen egyenértékű az F specifikációval (lásd a *b)* alpontot).

d) Valós számkonverzió előírt számú decimális jegyre. A G specifikáció

Az előző két alpontban megismertük a fixpontos és a decimális lebegőpontos konverziós lehetőségeket. A jelen alpontban bemutatandó konverziótípus bizonyos átmenetet jelent az előző kettő között.

Láttuk a *b)* alpontban, hogy fixpontos F konverzió esetében a *d* számmal a tizedespont utáni decimális jegyek szá-

mát specifikáljuk, míg az egész jegyek számát lényegében a mezőszélességgel adjuk meg. Ebből következik, hogy a specifikáció csak akkor alkalmazható a túl- és alulcsordulás veszélye nélkül, ha a kiíratandó mennyiség nagyságrendje nem ingadozik túlságosan nagy határok között. Lényeges, hogy a kiíratandó mennyiség abszolút értékének egy jó felső becslését ismerjük, mert enélkül nem tudjuk helyesen megválasztani a mezőszélességet.

A fenti nehézségek az E specifikáció kapcsán nem merülnek fel, viszont a tapasztalat szerint a lebegőpontos formátumban kiíratott számok vizuálisan valamivel nehezebben áttekinthetők, mint a fixpontosak. Ezt a két hátrányos tulajdonságot kívánja optimálisan kiküszöbölni a G specifikáció. Tekintettel arra, hogy input esetén az F és E specifikációk hatása azonos, ettől a G specifikáció hatása sem tér el. A továbbiakban tehát csak az output esetével foglalkozunk.

Output G specifikációval

A G specifikációval kiíratott számok általában nem tartalmaznak decimális exponenst, de itt nem a tizedespont utáni, hanem az *értékes decimális jegyek számát írjuk elő.* A gép a kiíratandó számot d számú értékes decimális jegyre íratja ki, miközben a tizedespont oda kerül, ahová a szám nagyságrendje megkívánja.

A G specifikáció alakja:

$$\mathrm{G}w\cdot d \qquad (6.61)$$

ahol w a mezőszélesség, d pedig az értékes decimális jegyek száma. Felhívjuk a figyelmet arra, hogy *d értelmezésében a G specifikáció eltér az általános megállapodástól.* A kiíratás eredményeként megjelenő numerikus mező jobb szélén 4 betűköz karakter áll, amelyek a túl- vagy alulcsordulás esetére fenntartott karakterpozíciók helyét töltik ki (lásd alább).

Példák

1. Legyen a pillanatnyilag feldolgozott FORMAT specifikáció

$$\mathrm{G}13\cdot 6 \qquad (6.62)$$

A következő táblázat a kiíratandó változó különböző értékeire azoknak az output rekordon megjelenő képét mutatja be:

A változó értéke:	*A kiírt alak:*	
−0.445862	−0.445862▽▽▽▽	
1.35468×10^3	▽▽1354.68▽▽▽▽	(6.63)
−22.6458998	▽ −22.6459▽▽▽▽	
128907	▽▽128907.▽▽▽▽	

A gép tehát mindig 6 értékes jegyre nyomtat, a tizedespont pedig a kinyomtatandó szám nagyságrendjétől függően „vándorol" a számon belül. Valódi tizedestörtek esetén a gép még egy zérus egész részt is nyomtat, akárcsak az F specifikáció esetében (lásd az első számot a táblázatban).

Azonnal felvetődik a kérdés, mi történik abban az esetben, ha a szám nagyságrendje olyan, hogy a tizedespont „vándorlása" közben a számon kívülre kerülne? Példánknál maradva, hogyan nyomtatná ki a gép az adott specifikáció mellett a 12 890 700, vagy a −0.00445862 számokat? Erre a kérdésre a feleletet a következő szemantikai szabály adja meg: *Ha a kiíratandó mennyiség nagyságrendje miatt a tizedespont a kiírt számon kívülre esne, akkor automatikusan át kell térni a lebegőpontos kiírási módra, azaz ilyen esetben a* Gw•d *specifikáció egyenértékű az* Ew•d *specifikációval.* (A tizedespont — mint a (6.63) táblázat mutatja — eshet a szám jobb vagy bal szélére.) A számok jobb szélén fenntartott betűköz karakterek ilyen esetben az E specifikációnak megfelelő decimális exponens elhelyezésére szolgálnak. A két példaként adott szám tehát a következő formátum szerint nyomtatódik:

A változó értéke:	*A kiírt alak:*	
12 890 700	▽0.128907E 08	
−0.00445862	−0.445862E−02	(6.64)

2. Legyen A tízelemű valós vektor; egyes elemeinek aktuális értékei legyenek a következők:

$A_1 = 23.7788543$ $A_6 = 0.00000673500765$

$A_2 = 0.00599607554$ $A_7 = 56.7543928$

$A_3 = 157.632544$ $A_8 = 9076238900.74$

$A_4 = -0.0768905332$ $A_9 = -0.3452877907$

$A_5 = 12490356.805$ $A_{10} = -0.000000764999802$

Tekintsük a következő FORTRAN programrészletet:

```
      DO 2 I=1,10
2     WRITE (5,9) I, A(I), A(I), A(I)                    (6.65)
9     FORMAT (I5, E15.6, F15.6, G15.6)
```

Feltesszük, hogy az 5-ös logikai perifériaszám a sornyomtatót jelenti. Mi történik a (6.65) program hatására, és milyen nyomtatási kép jön létre az átvitelek elvégzése után?

Megoldás. Könnyű látni, hogy a program az A vektor elemeit nyomtattatja ki, mégpedig — sorszámozott sorokban — minden elemet háromszor, egyszer E, egyszer F és egyszer G specifikáció mellett. A program tehát a különböző valós konverziós specifikáció összehasonlításához készít illusztratív nyomtatási képet. Az eredménylap a következő:

```
 1   0.237789E 02      23.778854▽▽▽▽23.7789▽▽▽▽
2▽▽▽0.599608E-02       0.005996▽▽▽0.599608E-02
 3   0.157633E 03     157.632544  ▽▽157.633▽▽▽▽
 4  -0.768905E-01      -0.076891  -0.768905E-01
 5   0.124904E 08▽??????????????  ▽0.124904E 08
 6   0.673501E-05       0.000007  ▽0.673501E-05
 7   0.567544E 02      56.754393  ▽▽56.7544▽▽▽▽
 8   0.907624E 10▽??????????????  ▽0.907624E 10
 9  -0.345288E 00      -0.345288  -0.345288▽▽▽▽
10  -0.765000E-06       0.000001  -0.765000E-06
```

A (6.66) táblázaton jól összehasonlítható a háromféle konverziós specifikáció az előforduló különböző nyomtatási esetekben. Vegyük sorra ezeket!

Az E specifikációnak megfelelő oszlopnak nincs különös érdekessége — ennek a specifikációnak éppen az az egyik legnagyobb előnye, hogy nincsenek különféle esetei. Legfeljebb annyit jegyzünk meg, hogy a mantisszát ott, ahol erre szükség volt, 6 jegyre kerekítettük. (Ezt egyébként a másik két specifikációnál is több helyen meg kellett tenni.)

A következő — az F specifikációnak megfelelő — oszlopban két helyen: az ötödik és a nyolcadik tömbelemnél, túlcsordulás történt; az aktuális értékek az adott specifikáció esetén nem nyomtathatók ki. Ezt a körülményt a szám helyére tett kérdőjelekkel szimbolizáltuk, azonban ez nem jelenti azt, hogy minden gépi reprezentációban ilyen módon diagnosztizálják a kiírási („külső") túlcsordulást. Szokásos még túlcsordulás esetén az áttérés E specifikációira, vagy a fixpontos kinyomtatás megtartása, mely esetben a nyomtatási formátum természetesen felborul. Ugyancsak említést érdemel még a hatodik és a tizedik elem, melyek nagysága a fixpontos alulcsordulás határán mozog; a tizedik elemből nem is maradt egyéb, mint egy kerekítésből származó egyes a legutolsó helyi értéken. Itt a konverzió folyamán minden értékes információ „elcsorgott".

Legváltozatosabb a G specifikáció kiírási oszlopa. Mindenekelőtt szembetűnő, hogy a 2., 4., 5., 6., 8. és 10. tömbelemeknél a specifikáció átment az E specifikációba, ami miatt a nyomtatási kép mindennek mondható, csak tetszetősnek nem. Figyeljük meg azt is, hogy ha a kinyomtatott szám zérussal és tizedesponttal kezdődik — ez a helyzet, ha a specifikáció átmegy az E specifikációba, de akkor is, ha a kiírandó szám 0.1 és 1 közé esik —, a szám egy karakterpozícióval kijebb kezdődik, mint ahogyan az 1-nél nagyobb kiírt számok kezdődnek. Hasonlítsuk össze az 1., 3., és 7. elemeket a többivel!

Milyen következtetéseket vonhatunk le a (6.66) táblázatból?

1. Mindenekelőtt le kell szögeznünk, hogy ha a kinyomtatandó mennyiségek nagyságrendje olyan tág határok között ingadozik, mint a jelen példában, akkor más kiírási specifikáció, mint a lebegőpontos (E), szóba sem jöhet. 14 decimális

nagyságrend átfogására sem az F, sem a G specifikáció nem alkalmas.

2. Az F specifikáció alkalmazásának javallatairól korábban már szóltunk, most csak utalunk arra, hogy használata során túl-, és alulcsordulási veszély lép fel, tehát csak jól ismert nagyságrendű mennyiségek kiíratására célszerű használni (pl. bemenő adatok ellenőrzési célból történő visszaíratása).

3. Ami a G specifikációt illeti, ezt akkor ajánlhatjuk, ha a kiíratandó mennyiségekről tudjuk, hogy abszolút értékben 1 és 10^d közé esnek. Ebben az esetben az előállított számoszlopok mindkét széle egyenes, míg az F specifikáció olyan számoszlopokat állít elő, melyek jobb széle egyenes, bal széle általában cikcakkos. (A G konverzió előnye a (6.66) táblázatból nem tűnik ki ugyan, de ott a számok nem is elégítik ki az említett nagyságrendi korlátokat.)

Felhívjuk az olvasó figyelmét arra is, nem szabad azt várnia, hogy egy konverziós specifikáció több értékes jegyet adhat, mint amennyi a gépi számábrázolás véges pontosságából eleve adódik. Ha tehát valamely gép a valós mennyiségeket 11 decimális jegyre ábrázolja, akkor hiába íratjuk ki valamelyik valós változóját pl. E20.13 specifikációval, a mantissza kiírt jegyei közül az utolsó néhány biztosan teljesen értéktelen. Nem szabad megfeledkeznünk arról sem, hogy a számrendszerváltási algoritmus maga is bizonyos kerekítési hibával járhat, így a kiírt számok utolsó jegyének helyességében mindenképpen kételkednünk kell. Ez mindegyik konverziós specifikációra egyaránt érvényes.

e) Kétszeres pontosságú mennyiségek átvitele. A D specifikáció

Kétszeres pontosságú (vagyis DOUBLE PRECISION típusú) mennyiségek átvitelére a D konverziós specifikációt használhatjuk, melynek alakja:

$$\mathrm{D}w.d \tag{6.67}$$

ahol *d* és *w* jelentése az előzőkből ismert.

A D specifikáció teljesen azonosan viselkedik az E specifikációval, csak megfelelő I/O listaelemnek DOUBLE PRECISION

típusúnak kell lennie, és kinyomtatáskor az eredménylapon az exponenst nem E, hanem D betű jelzi.

A D specifikációval beolvastatott numerikus mezőnek nem kell feltétlenül „D betűs", decimális normálalakú számnak lennie; lehet bármiféle decimális numerikus mező, akárcsak az E, F vagy G specifikáció esetén. A beolvasott szám kétszeres pontosságú lebegőpontos ábrázolásúra konvertálódik, és elhelyeződik az I/O listaelemnek megfelelő változóban. (Valójában az input rekord decimális konstansaiban a D betűnek semmiféle jelentősége nincs, mert azt, hogy egy numerikus mezőt egyszeres vagy kétszeres pontosságúra konvertálunk, nem a D betű jelenléte vagy jelen nem léte dönti el, hanem az I/O listaelem típusa.)

Az ASA FORTRAN nem határozza meg, hogy mi történik, ha kétszeres pontosságú mennyiségeket egyszeres pontosságú specifikációk mellett íratnak ki. Ez általában nem hiba, de a kiírt szám csak annyi értékes jegyet tartalmaz, amennyi az egyszeres pontosságú számábrázolásnak megfelel.

f) Logikai mennyiségek átvitele. Az L specifikáció

Logikai változók átvitelére az L konverziós specifikációt használjuk, melynek alakja

L*w*

A specifikációnak megfelelő I/O listaelemnek logikai (LOGICAL típusú) változónak kell lennie.

Ha ezt a specifikációt *kiviteli* műveletben alkalmazzuk, akkor a kiírt output mező első $w-1$ karaktere betűköz karakter, utolsó karaktere pedig egy T vagy egy F betű, aszerint, hogy a változó aktuális értéke TRUE vagy FALSE.

Beolvasási műveletekben az L specifikációnak megfelelő karaktermezőt esetleg tetszőleges számú betűköz karakter vezetheti be, majd egy T vagy F betűvel kezdődő tetszőleges jelsorozatnak kell következnie. Ha a jelsorozat T betűvel kezdődik, akkor a beolvasandó logikai változó a TRUE értéket veszi fel, ha pedig F betűvel, akkor a FALSE értéket.

Itt felhívjuk a figyelmet egy kis következetlenségre. A numerikus konverziós specifikációknál megszoktuk — és ez természetes is —, hogy

az egyes kiviteli konverziók eredményeként az output rekordon egy, az ASA FORTRAN szintaxisának megfelelő numerikus mező jelenik meg. Hasonlóan, ha az input rekordon egy a „konstans" szintaktikus kategóriába tartozó jelsorozat áll, akkor ez a megfelelő specifikációk segítségével beolvastatható és konvertálható. Ez az L konverzióra nem teljesül, mert a logikai konstansok alakja .TRUE., ill. .FALSE., a kivitel során pedig nem ezek keletkeznek az output rekordon, és beolvasáskor sem fogadhatók el logikai bemenő adatként.

Egyes gépi reprezentációkban nem kötik ki, hogy az input mező T vagy F betűvel kezdődjék, csupán annyit írnak elő, hogy vagy T betűvel, vagy bármi mással. Ha az input mező T betűvel kezdődik, akkor a változó a TRUE értéket veszi fel, ellenkező esetben a FALSE értéket. Mint érdekességet jegyezzük meg, hogy ha egy ilyen gépi reprezentációban, az input rekordban akár a .TRUE., akár a .FALSE. jelsorozat fordul elő, a beolvasás eredménye mindenképpen FALSE, hiszen az első karakter nem T betű, ezért a gép a mező értékét FALSE-ként veszi figyelembe.

g) Szöveg beolvasása és kiírása. Az A specifikáció

Adatfeldolgozási feladatokkal kapcsolatban gyakran célszerű a program minden bemenő adatkötegét kiegészíteni valamilyen bevezető karaktersorozattal, melynek segítségével az eredménylapon azonosíthatjuk a különböző adatkötegekhez tartozó eredményeket. Különösen fontos ez akkor, ha a bemenő adatsorozatok száma igen nagy, és egy futtatási menet után nagyszámú, de egymástól kevéssé különböző eredményt kapunk. Ez a helyzet pl. méréskiértékelő programok esetében. Ha ugyanaz a program egyszerre dolgoz fel több — esetleg különböző forrásból származó — adatsorozatot, az eredmények azonosítása el sem képzelhető jól megválasztott informatív szövegek nélkül. Ezek tartalmazhatják az adatsorozat sorszámát (vagy valamely egyéb kódszámát), az adatfelvételezés dátumát, helyét, a felvételező személy nevét stb.

A bevált programozási technika az, hogy ezeket az azonosító szövegeket a bemenő adatokkal együtt beolvastatjuk, majd az eredménylapra változtatás nélkül kiíratjuk. Szükségünk van tehát valamilyen átviteli specifikációra, amelyet elsősorban akkor alkalmazunk, ha szövegeket kívánunk előbb beolvastatni, majd (ugyanezen specifikációval) később kiíratni. Erre való az A specifikáció, melynek alakja

$$\mathrm{A}w \qquad (6.68)$$

és ahol *w* jelentése, mint a következőkben látni fogjuk, eltér megszokott értelmezésétől.

Annak érdekében, hogy szabatosan kifejthessük az A specifikációval kapcsolatos szabályokat, bevezetünk egy újabb fogalmat és egy jelölést. A mezőszélesség mintájára, *elemszélességnek* nevezzük és *g*-vel jelöljük az I/O lista pillanatnyilag feldolgozás alatt álló elemében tárolható karakterek számát. Emlékeztetünk rá — lásd a Negyedik fejezet *5a)* alpontját —, hogy bármilyen típusú változó tartalmazhat numerikus információ helyett karakterinformációt is. Hogy ez esetben hány karaktert tartalmazhat, az függ a konkrét gépi reprezentációtól, és általában a változó típusától is. A *g* szám a pillanatnyi I/O listaelemben tárolható karakterek számát jelenti, tehát arra a változóra vonatkozik, amelyiknek tartalmát éppen átvisszük.

Mielőtt az A specifikáció szemantikájába mélyebben belemennénk, tekintsük azt a gyakorlatilag legfontosabb esetet, amikor az elem- és a mezőszélesség megegyezik, azaz $w=g$. *Kiviteli* utasítás esetén egy ilyen specifikáció hatására a változóban tárolt, *g* számú karakterből álló szöveg kiíródik az output rekordra, és azon egy *g* szélességű karaktermezőt hoz létre. *Beviteli* utasításban az input rekord soronkövetkező *w* karaktere beolvasódik, elhelyeződik az aktuális I/O listaelemben, és — $w=g$ lévén — ezzel éppen kitölti annak valamennyi karakterpozícióját.

Példa

Tegyük fel, hogy valamely gépi reprezentációban a valós változókra $g=8$. Írjunk beolvasási utasítást, amelynek segítségével beolvashatjuk egy 80 oszlopos lyukkártya valamennyi karakterpozícióját!

Megoldás. Minthogy 80 karaktert kell beolvastatnunk, ezek tárolására 10 valós változóra van szükségünk. Legcélszerűbb egy tízelemű vektort definiálnunk, amely éppen 80 karakterpozíciót tartalmaz. A beolvasó utasítások:

```
      DIMENSION R(10)
      ....................
      READ (2,12) R                    (6.69)
   12 FORMAT (10A8)
```

Az itt fellépő READ utasítás I/O listája egyetlen tömbelemből áll, amely — mint ismeretes — a tömb valamennyi elemének beolvasását jelenti. A FORMAT utasításban az A specifikáció előtti 10-es ismétlési tényező éppen azt írja elő, hogy minden egyes tömbelemet egyaránt A8 specifikációval kell beolvasni.

Javasoljuk az olvasónak, hogy a (6.69) példát vésse emlékezetébe, mint a lyukkártyáról történő karakterbeolvasás „iskolapéldáját". Már e helyen felhívjuk a figyelmet, hogy (6.69)-ben nem írható a 12-tes címkéjű FORMAT utasítás helyett

```
12    | |FORMAT (A80)
```
(6.70)

Jelenlegi ismereteink nem elégségesek ennek az egyébként roppant tanulságos hibának a pontos elemzésére, így erre az 5. feladatban térünk majd vissza.

Ha a (6.69) utasításokkal beolvastatott szöveget a program egy másik helyén a sornyomtató első 80 karakterpozíciójára ki akarjuk íratni, akkor egyszerűen a

```
WRITE (5,12) R
```
(6.71)

utasítást kell alkalmaznunk.

Ezzel letárgyaltuk a $w=g$ alapesetet. Következnek a $w>g$ és a $w<g$ esetek.

A $w>g$ esetben, tehát amikor a mezőszélesség nagyobb, mint az elemszélesség, a beolvasandó input mező nem fér el teljes egészében az I/O listaelemben. Az idevonatkozó szintaktikus szabály:

Ha az input karaktermező szélessége meghaladja az I/O listaelem szélességét, akkor a gép az input mező utolsó g számú karakterét helyezi el a megfelelő változóban, az ezt megelőző karaktereket pedig nem veszi figyelembe. A kiírás esetére szóló szabály pedig:

Ha a kiíratandó változó elemszélessége kisebb, mint az output mező specifikált szélessége, akkor a változóban tárolt karakterek az output mező utolsó g karakterét foglalják el, az ezt megelőző karakterek pedig betűközökkel töltődnek fel.

Példa

Legyen X valós változó, és tegyük fel, hogy az adott gépi reprezentációban a valós változók 8 karaktert tárolhat-

nak. Legyen a 11 karakter szélességű input karaktermező a következő:

HOVANSCSINA

Olvastassuk be, és írassuk ki ezt a karaktermezőt a következő utasításokkal:

```
      READ (2,7) X
      WRITE (5,7) X
    7 FORMAT (A11)
```
(6.72)

Mi lesz a (6.72) utasítások végrehajtása után az X változó tartalma, és a kiírt karaktermező?

Megoldás. A két előző szabály szerint, mivel a beolvasáskor az input mező utolsó 8 karakterét vehetjük figyelembe, az X változó tartalma a beolvasás után:

ANSCSINA

Ennek a nyolc karakternek az A11 specifikáció szerinti kiíratásával a következő output mezőt kapjuk:

▽▽▽ANSCSINA

Látható, hogy a $w>g$ esetben lényegében a jobbról tömörítési elv érvényesül, amit a numerikus konverzióknál már megszoktunk. (A később tárgyalandó $w<g$ esetben viszont a balról tömörítésre látunk majd egy példát.)

A (6.72) példa kapcsán nem árt felhívni a figyelmet arra a körülményre, hogy a $w>g$ specifikáció melletti beolvasás és ugyanezen specifikációval való visszaíratás közben általában $w-g$ számú karaktert elveszítünk. Ez persze nem feltétlenül baj, sőt, bizonyos esetekben ki is használhatjuk fölösleges karakterek ignorálására.

A $w<g$ esetben az input karaktermező nem tölti ki az aktuális I/O listaelemet. A megfelelő szemantikai szabály most balról (és nem jobbról!) tömörítést ír elő:

Ha az input karaktermező szélessége nem éri el az I/O listaelem szélességét, akkor a gép az input mező karaktereit a változó első w karakterpozíciójában helyezi el, a fennmaradó $w-g$ számú karakterpozíció pedig betűköz karakterekkel töltődik fel. A kiírás esetére szóló szabály a következő:

Ha a kiíratandó változó elemszélessége meghaladja az output mező specifikált szélességét, akkor a kiíratandó változó első w karaktere íródik ki, a fennmaradó w − g számú karakter nem vesz részt az átvitelben.

A szabályokból következik, hogy ha valamilyen karakterinformációt az elemszélességnél kisebb szélességű mezőkre bontunk, és úgy olvastatjuk be és íratjuk vissza, akkor nem vesztünk információt, csak a szükségesnél több memóriarekeszt kötünk le a szöveg tárolására. A (6.69) példa eszerint így módosítható:

```
      DIMENSION R(20)
      ......................
      READ (2,12) R                    (6.73)
12    FORMAT (20A4)
```

Ha a (6.73)-mal beolvastatott szöveget egy

```
      WRITE (5,12) R
```

utasítással visszaíratjuk, az eredmény pontosan ugyanaz, mint a (6.69)—(6.71) utasítások esetében.

Példa

Legyen ismét a valós típusú változókra $g=8$; legyen R háromelemű vektor, és a 18 karakterből álló input karaktermező tartalmazza a következő jelsorozatot:

SANDORFERENCISTVAN

Olvastassuk be és írassuk ki ezt a karaktermezőt a következő utasításokkal:

```
      READ (2,10) R
      WRITE (5,12) R                   (6.74)
10    FORMAT (3A6)
12    FORMAT (3A8)
```

Mi a (6.74) program hatása?

Megoldás: a beolvasási utasításhoz tartozó formátumban $w < g$, így az R vektor elemeiben a beolvasott információ ekként helyezkedik el:

R(1): SANDOR▽▽
R(2): FERENC▽▽
R(3): ISTVAN▽▽

A vektorelemek tartalmának kiíratása után — minthogy kiíráskor mind a nyolc karakterpozíciót figyelembe vesszük — a következő adódik:

SANDOR▽▽FERENC▽▽ISTVAN

A tárgyalt szemantikai szabályokat tehát arra használtuk ki, hogy betűköz karaktereket helyezzünk az egyes szavak közé.

h) Szöveg beolvasása és kiírása. A H specifikáció

Szövegek beolvasására és kiíratására az előző alpontban megismert lehetőségeken kívül használható egy további átviteli specifikáció is, melyet a H betűvel jelölünk.

A H specifikáció, ellentétben az A specifikációval, a szerkesztési specifikációk kategóriájába tartozik. Mint már említettük — lásd a 4. pont bevezető részét —, a konverziós és a szerkesztési specifikációk között az a különbség, hogy az utóbbiakhoz nem tartozik az I/O listának eleme.

A H specifikáció alakilag Hollerith konstans:

$$wHXX\ldots X \tag{6.75}$$

ahol az X-ek a FORTRAN jelkészletbe tartozó tetszőleges alfanumerikus karaktereket jelölnek.

Gépi reprezentációkban általában megengednek a FORTRAN-karaktereken kívül a bemenő egység jelkészletébe tartozó egyéb karaktereket is.

Kiírási utasításokban a H specifikáció maga határozza meg a kiírandó karaktereket: hatására a H betű után álló *w*

számú karakter kiíródik az output rekordra. A H specifikációnak beolvasási utasításokkal kapcsolatos szerepére később térünk vissza.

A H specifikációt magyarázó szövegek, fejlécek, feliratok kinyomtatására használjuk. Használatára nemcsak ebben az alpontban, hanem a könyv további részeiben is bőven látunk majd példákat.

Példák

1. Írjunk FORMAT utasítást a következő szöveg kiírására:

A PROGRAM EREDMENYEI

Megoldás. A kiírandó szöveg 20 karakterből áll, tehát a specifikációban $w = 20$. A megfelelő kiírási utasítások:

```
      WRITE (5,12)                                (6.76)
12    FORMAT (20HA▽PROGRAM▽EREDMENYEI)
```

(6.76) érdekessége, hogy a WRITE utasításnak egyáltalában nincs I/O listája. Bináris átvitelek esetében ez egy üres rekord kiírását jelentette (lásd 206., old., 2. szabály), karakterátvitelek esetében nem feltétlenül. A (6.76) utasításpárban sem üres a kiírt rekord, hanem a példa által előírt 20 karakterből álló szöveget tartalmazza. Ezt a szöveget *magában* a FORMAT *utasításban adtuk meg*.

2. Legyen VAL egy valós típusú tömb, melynek 10 sora és három oszlopa van. Jelentsék a tömb egyes sorai bizonyos árucikkek árát dollárban, svájci frankban és nyugatnémet márkában. Feltesszük, hogy olyan árucikkekről van szó, amelyeknek ára egyik pénznemben sem éri el a százezret. Az árakat két tizedes pontosságig kell kiíratni. Minden számadat elé nyomtassuk ki a megfelelő pénznem rövidítését (dollár = $, svájci frank = SFR, márka = DM)*. A táblázat lapközépre állítását ezúttal nem követeljük meg, mindössze 5 karakter szélességű bal margót írunk elő. Írjuk fel a megfelelő kiíró utasításokat!

* Feltesszük, hogy olyan gépi reprezentációban dolgozunk, amely megengedi a H specifikációkban dollárjel használatát.

Megoldás. Ismert negységrendű valós változókról lévén szó, az F specifikációt célszerű használni, mégpedig a két tizedes pontosságra való tekintettel $d=2$ választással. Ötjegyű egész résszel kell számolnunk, tehát a *b)* alpontban mondottak szerint a mezőszélesség (legalább) $w=9$ kell legyen. Maradjunk tehát az F9.2 specifikációnál!

A bal margó és az elválasztó betűközök problémáját ezúttal úgy oldjuk meg, hogy minden egyes szám elé H specifikációval szövegként kinyomtatunk öt betűköz karaktert, és ezt követően a megfelelő valutarövidítést. Magát a programot a már ismert fogások felhasználásával állítottuk össze. A megoldás a következő:

```
  |  |DO 2 I=1, 10
 2|  |WRITE (5,8)(VAL(I,J),J=1,3)
 8|  |FORMAT (6H▽▽▽▽▽$, F9.2,                      (6.77)
  | X|8H▽▽▽▽▽SFR, F9.2, 7H▽▽▽▽▽DM, F9.2)
```

A példából láthatjuk, hogy betűközöket H specifikációval is nyomtathatunk, bár nagyobb számú betűköz esetén ez nem kényelmes és van egyéb módja is (lásd a következő alpontot).

Ismételten emlékeztetünk arra, hogy a H specifikációnak az I/O listán nem felel meg elem; az I/O lista (az összetett listaelemet dinamikusan értelmezve) tehát 3 elem átvitelét specifikálja, a FORMAT lista pedig 6 elemből áll.

A H specifikációt alkalmazhatjuk *beolvasási* utasításokban is. Rendeltetése hasonló az A specifikációéhoz: Hollerith mezők beolvastatása, későbbi visszaíratás céljából. (A FORTRAN felhasználók nagy része ezt a lehetőséget nem ismeri és ezért nem is használja, pedig sok esetben igen jó szolgálatot tehet.)

Az ASA FORTRAN definíciói szerint *ha egy* READ *utasításhoz rendelt* FORMAT *utasításban* H *specifikáció fordul elő, akkor ez karakterbeolvasást jelent magába a* FORMAT *utasításba.*

„Beolvasás magába a FORMAT utasításba" — ezt kissé közelebbről meg kell világítanunk. Arról van szó, hogy ha valamely szöveget H specifikációval beolvastatunk egy adott FORMAT utasításba, majd később ugyanezen FORMAT uta-

sítás szerint kiíratunk valamit, akkor ezen későbbi kiíratáskor a FORMAT úgy fog viselkedni, mintha a megfelelő H specifikációba az imént beolvasott szöveg volna beírva. Gyakorlatilag ezt úgy alkalmazzuk, hogy amikor írjuk a programot, a H specifikációba betűközöket írunk be, majd a specifikációt mintegy „kitöltjük" az input rekordról beolvasott Hollerith mezővel.

Példák

1. Tegyük fel, hogy valamilyen adatfeldolgozási programban a számadatokat egy legfeljebb 20 karakterből álló Hollerith mező előzi meg, amely kizárólag eredmény-azonosítási célokat szolgál. Minden egyes adatfeldolgozási menet ennek az információnak a beolvastatásával és kiíratásával kezdődik. Az egyes adatfeldolgozási meneteket a következő utasításokkal kezdhetjük:

```
      READ (2,8)
      WRITE (2,8)                                   (6.78)
8     FORMAT(20H▽▽▽▽▽▽▽▽▽▽▽▽▽▽▽▽▽▽▽▽)
      ......
```

A READ utasítás beolvassa, a WRITE pedig visszaírja a beolvasott Hollerith mezőt. Ha pl. a (6.78) utasítások első végrehajtásakor a bemenő adatok előtt az ELSO▽ADATSOROZAT▽▽▽▽ szöveget helyeztük el, akkor az első WRITE ezt a szöveget fogja visszaíratni. Amikor a soron következő ciklusban visszatérünk a READ utasításra, és beolvassuk a következő szöveget, amely mondjuk MASODIK▽ADATSOROZAT▽, akkor a WRITE már ez utóbbit íratja ki, és így tovább.

2. A (6.69)—(6.71) utasítások kapcsán felvett feladatot kissé általánosítva, írjunk FORTRAN programot, amely lyukkártya kötegek sornyomtatón való kiíratására alkalmas! Tegyük fel, hogy a kiíratandó lyukkártya köteg elé egy paraméterkártyát helyezünk el, amelynek első öt oszlopában álló egész szám azt adja meg, hogy összesen hány lyukkártyát kell kiíratnunk. Legyen a lyukkártya-olvasó logikai perifériaszáma 2, a sornyomtatóé 5.

Megoldás. A feladatnak megfelelő program a következő:

```
      MASTER LIST
      READ(2,4) N
      DO 1 I=1,N
      READ(2,5)
1     WRITE(5,5)
      STOP
4     FORMAT(I5)                                          (6.79)
                        56 betűköz karakter
5     FORMAT(80H...  ___________________________ ...
     X...________________________...)
              24 betűköz karakter
      END
```

Azt hisszük, a program nem szorul sok magyarázatra. Annyit szükségesnek tartunk megjegyezni, hogy az 5-ös címkéjű FORMAT utasításban a 80 betűköz nem fér ki egy sorban, hiszen a lyukkártyán a program szempontjából csak 72 hasznosítható karakterpozíció van. Így tehát a FORMAT utasítást egy folytatósorban kellett folytatni. Hollerith-konstansoknak folytatósorban történő folytatásakor azonban bizonyos óvatossággal kell eljárnunk. Gondosan ügyelni kell ugyanis arra, hogy a *programsort teljesen végigírjuk a* 72. *karakterpozícióig bezárólag*. Egyéb utasítások esetén — mint tudjuk — ez nem lényeges, bárhol abbahagyhatjuk a sort, és kitéve a folytatás jelzését, áttérhetünk egy újabb sorra. Itt nem. A gép a beolvasáskor ugyanis automatikusan a 72. karakterpozícióig számítja a Hollerith-konstanst, és ha az nem terjed eddig, akkor a ki nem használt pozíciókat betűköznek tekinti, és beleszámítja a konstansba. Ennek következtében egyrészt nem kívánt betűköz karakterek kerülnek a konstansba, másrészt a mezőszélességgel baj lesz, mert a beolvasott (észlelt) karakterek száma nem egyezik meg a karakterek kívánt számával. Ez általában végrehajtási hibát okoz a program futásakor, mert a

gép nem találja meg a Hollerith-konstans utáni FORMAT listaelemet. Ezeket a szempontokat az 5-ös FORMAT utasítás felírásakor is figyelembe kell venni, ha a programot egy konkrét gép számára kívánjuk lelyukasztani. Még annyit, hogy a fentiek ugyan főként lyukkártyás bemenet esetére vonatkoznak, de az elmondottakat lyukszalagos bemenet esetén is szem előtt kell tartani, mert ha nem írjuk végig a 72 karakterpozíciót, akkor a sort lezáró „új sor" karaktert a gép esetleg a Hollerith-mező részeként tekinti, ami szintén zavart okozhat.

Több gépi reprezentációban a H specifikációnak egy igen hasznos és kényelmesen alkalmazható változata van. Ebben a változatban a specifikációban elmarad a mezőszélességet jelző *w* szám és a H betű; ezek helyett a Hollerith mezőt egyszerűen idézőjelbe tehetjük. (Idézőjel helyett egyes gépeken jelkészlettől függően valamely egyéb, megállapodásszerűen rögzített karakter is szerepelhet.) Tehát a 6HSANDOR specifikáció így is írható: 'SANDOR'. Ez azért nagyon kényelmes, mert a programozónak nem kell megszámlálnia a Hollerith konstans karaktereit.

i) Helykitöltés. Az X specifikáció

Betűközöknek az output rekordra való kiírására már eddig három lehetőséget ismertünk meg. Betűköz karakterekkel tölthetjük fel az output numerikus mezők bal szélső karaktereit, ha a mezőszélességet nagyobbra választjuk, mint amennyire a szám kiírásához feltétlenül szükség van. Erre az *a)*—*f)* alpontokban láttunk példákat. A másik lehetőség betűközök kiírására a H specifikáció, amelyben a betűközöket „szövegként" adhatjuk meg (lásd pl. a (6.78) és (6.79) programrészleteket). Kiírathatunk betűközöket ezenkívül A specifikációval is (*g)* alpont).

Mindezekben a specifikáció típusokban azonban a betűközök kiírása csak mintegy „melléktermékként" adódik. Most megismerkedünk azzal a szerkesztési specifikációval, amely kifejezetten erre a célra szolgál, és amelyet beolvasási utasításokban az input mező bizonyos karaktereinek ignorálására használhatunk fel.

E specifikáció alakja

$$wX \qquad (6.80)$$

Kiírási utasításokban ez *w* számú betűköz karakter kiírását jelenti az output rekordra. *Beolvasás* esetén a gép figyelmen

kívül hagyja (átugorja) az input rekord soron következő *w* karakterét.

Példák

1. Első példaként az X specifikáció alkalmazására térjünk vissza a (6.60) programhoz (232. old.). A feladatot egészítsük ki azzal, hogy írjuk elő cím és fejléc nyomtatását az eredménylap tetejére. A program címe legyen FUGGVENYTABELLAZAS, ezt írassuk ki középre állítva, majd mindkét kinyomtatott számoszlop közepe fölé nyomtassunk fejlécet, amely a független változó esetében az egyetlen X betűből, a függő változó esetében pedig az F(X) karakterekből álljon. Alakítsuk át ezenkívül a 3-as címkéjű FORMAT utasítást az X specifikáció alkalmazásával.

Megoldás. A módosított program a következő:

```
      MASTER TABL
      WRITE (5,4)
      WRITE (5,5)
      DO 2 I=1,1000                                  (6.81)
      ........... (a (6.60) program további utasításai)
   3  FORMAT (48X, F6.3, 5X, E13.6)
   4  FORMAT (51X, 18HFUGGVENYTABELLAZAS)
   5  FORMAT (51X, 1HX, 12X, 4HF(X))
      END
```

A (6.60) programhoz képest (6.81) két WRITE utasítással, valamint a hozzájuk tartozó 4-es és 5-ös FORMAT-tal bővült. A (6.81) program által előállított nyomtatási kép már valamivel tetszetősebb a (6.60) által előállítottnál, de még mindig „nem az igazi”. Igazán áttekinthetővé akkor tudjuk majd tenni a nyomtatási képet, ha megtanuljuk, hogyan kell üres sorokat kihagyni és új lapot nyitni. Az itt előírt formátum ugyanis „horizontálisan” már szépen tagolt, de „vertikálisan” még nem: a cím alatti sorban rögtön a fejléc, ez alatt pedig sorköz nélkül, azonnal az eredmények következnek. A végleges változatot majd (6.93) alatt látjuk.

2. Alkalmazzuk az X specifikációt beolvasási műveletekben! Tegyük fel, hogy az input rekordon valamely A és B mennyiségek vannak elhelyezve, a következő formátum szerint:

|▽▽▽▽▽A▽ =139.4077▽▽▽▽▽▽▽▽▽▽▽B▽ =1078.503

Olyan READ utasítást kell írnunk, amelynek segítségével A és B számértékét „leválaszthatjuk" az őket környező szövegről, és értéküket beolvashatjuk a megfelelő változóba.

Megjegyezzük, hogy ilyenfajta input rekord többnyire oly módon jön létre, hogy valamely korábbi program eredményeit lyukszalagra lyukasztjuk és ezt a lyukszalagot használja bemenetként a következő program. A bevitel szempontjából közömbös és ezért ignorálandó szövegek az előző program output formátumának „maradványai".

A fenti input rekordot a következő utasítással olvastathatjuk be:

```
      READ (2,8) A, B
                                              (6.82)
8     FORMAT (8X, F8.4, 13X, F8.3)
```

A FORMAT utasításban az X specifikációk biztosítják az input szempontjából közömbös szövegek és betűközök átugratását.

j) Új rekord megnyitása. A vessző és a ferde törtvonal mint elhatároló jelek

Eddigi ismereteink alapján egy FORMAT utasításban mindig csak egyetlen rekord formátumát tudjuk megadni: ha új rekordot kell nyitnunk (pl. soremelés), mindig új átviteli utasítást kell kezdenünk. Most megtanuljuk, hogyan kezdhetünk új rekordot átvitel közben, hogyan írhatunk ki teljesen üres rekordokat, és beolvasáskor hogyan ugrathatunk át teljes rekordokat.

Az új rekord megnyitását előíró jel a ferde törtvonal: /. Ez nem specifikáció, hanem olyan elhatároló jel, melynek önálló szemantikai jelentése is van. Elhatároló jelként pótolja a FORMAT lista egyes elemeit elválasztó vesszőt. Ha egy FORMAT listában két E specifikációt kell elkülönítenünk, akkor

ezt a formális szempontból teljesen egyenértékű következő alakok bármelyikével megtehetjük:

E13.6/E15.8 (6.83)

E13.6/,E15.8 (6.84)

E13.6,/,E15.8 (6.85)

Két specifikáció között nemcsak egy, hanem akárhány ferde törtvonal is állhat, azonban — a törtvonalnak konkrét jelentése lévén — azok száma nem választható meg tetszőlegesen. Szintaktikusan helyes pl. a következő felírásmód is:

E13.6///E15.8 (6.86)

azonban ennek jelentése eltér a (6.83)—(6.85) formák jelentésétől (a részletes szabályokat lásd alább).

A vesszőnek, mint elhatároló jelnek a használatáról még annyit kell tudnunk, hogy az *nem sokszorozható* (közvetlenül egymás után nem állhat több vessző), azaz (6.86)-tal ellentétben szintaktikusan hibás

E13.6, , , E15.8 (6.87)

A (6.83)—(6.85) példákhoz hasonlóan viszont — nem kötelezően — elhelyezhetők vesszők az egyes törtvonalak között:

E13.6//,/E15.8 (6.88)

E13.6,/,//,E15.8 (6.89)

stb. *A szintaktikusan helyesen elhelyezett vessző nem módosítja a* FORMAT *lista értelmét.*

Mint mondottuk, a törtvonal jelentése a FORMAT listában *új rekord megnyitása.* Kiírási (WRITE) utasítások esetében ez azt jelenti, hogy a kiírás a következő rekordban (azaz új sorban, vagy újabb kártyán stb.) folytatódik, beolvasási utasításokban pedig hatására a kurrens input rekord esetleg még hátralevő része ignorálódik, és a beolvasás a soronkövetkező rekord elejétől folytatódik. Több, egymás mellett elhelyezett törtvonal megfelelő számú üres rekord kiírását, ill. több rekord átugratását jelenti.

Példák

1. Térjünk ismét vissza a (6.60) programmal kapcsolatos feladathoz, és fejlesszük még tovább az output formátumot! A (6.81)-hez fűzött megjegyzésekben rámutattunk, hogy baj van még az eredmények „vertikális" tagoltságával. Bővítsük ki tehát az output formátumot üres sorokkal, amelyek elválasztják a címet a fejléctől, és a fejlécet a számszerű eredményektől. A cím és a fejléc közé öt, a fejléc és az eredmények közé három üres sort helyezzünk.

Megoldás. A feladat követelményeit könnyen kielégíthetjük (6.81) 4-es és 5-ös FORMAT utasításainak kis módosításával. A módosított utasítások:

```
4     | |FORMAT(51X, 18HFUGGVENYTABELLAZAS/////)
5     | |FORMAT(51X, 1HX, 12X, 4HF(X)///)            (6.90)
```

Ezzel a feladatot megoldottuk; a 4-es FORMAT utáni öt, ill. az 5-ös FORMAT utáni három törtvonal a megfelelő rekordok után öt, ill. három üres rekord kiírását (öt, ill. három soremelést) eredményez.

(6.90) tehát korrekt megoldása a feladatnak; az „új rekord" specifikáció segítségével azonban továbbmehetünk, és a (6.81) program két bevezető WRITE utasítását össze is olvaszthatjuk. Ennek megfelelően a 4-es és az 5-ös címkéjű FORMAT utasítás is egyetlen FORMAT-tá olvad össze. Adjuk az új FORMAT-nak a 4-es címkét, és nézzük meg, hogyan fest a két különálló utasítás együtt! A megfelelő utasítások:

```
 |  |WRITE(5,4)                                      (6.91)
4|  |FORMAT(51X, 18HFUGGVENYTABELLAZAS//////51X,
 | X|1HX, 12X,4HF(X)///)
```

Első ránézésre úgy tűnik, mintha a (6.91)-es FORMAT-ban egyszerűen egymás mellé helyeztük volna a (6.90) FORMAT-ok listáit, és ezzel a két rövidebb utasításból kaptunk volna egy hosszabbat. Ez lényegében így is van, egy dologra azonban fel kell hívnunk a figyelmet.

Mint láttuk, a (6.90)-ben szereplő 4-es címkéjű FORMAT listájának végén öt törtvonal-karakter áll, (6.91)-et tüzetesen

szemügyre véve viszont azt látjuk, hogy a két listarész között hat „új rekord" jel található.

Tekintettel arra, hogy az ismert szabály szerint az I/O lista vége egyben rekord vége is, az I/O lista kezdete pedig egyben rekord kezdet, (6.90)-ben a 4-es FORMAT lista végén csak öt „új rekord" jelet kellett elhelyeznünk, mivel az 5-ös FORMAT értelmezésének megkezdésekor a gép automatikusan új rekordot nyit. A cím kiírása után tehát ötször egymás után nyit (és azonnal bezár) egy-egy új rekordot, és az 5-ös FORMAT értelmezésének megkezdésekor nyitja meg a már nem üres hatodikat. Az eredmény tehát öt üres sor, a fejléc a hatodikba kerül.

A (6.91) megoldásban ezzel szemben az összes bevezető szöveget egyetlen WRITE-tal és egyetlen FORMAT-tal visszük át, ezért ahhoz, hogy a cím alatt öt üres sort kapjunk, a FORMAT lista közepén hat törtvonal karaktert kell elhelyeznünk. A FORMAT lista végén természetesen nem kellett megnövelnünk a törtvonalak számát.

Mindebből leszűrhetjük a következő szabályt: *Kiírási utasításokhoz rendelt* FORMAT *utasításban a* FORMAT *lista elején vagy végén elhelyezett k számú törtvonal k számú üres rekord, a* FORMAT *lista közepén elhelyezett k számú törtvonal k—1 számú üres rekord kiírását jelenti.* Speciálisan a FORMAT lista középen álló egyetlen törtvonal hatására a kiírás a soron következő rekordban folytatódik (soremelés).

2. Legyen R egy 15×15-ös valós mátrix. A feladat e mátrix elemeinek sorfolytonos kiíratása hatjegyű mantisszával, decimális normálalakban. A fenti pontosságú kiíratáshoz E13.6 specifikációt kell alkalmaznunk, és így egy mátrixsor nem fér el a sornyomtató egy sorában. Tördeljük tehát mindegyik mátrixsort három sornyomtató sorra! Annak érdekében, hogy az egyes mátrixsorok jól láthatóan el legyenek választva, mindegyik mátrixsor (azaz minden harmadik sornyomtató sor) után hagyjunk ki egy üres sort! Írjuk fel a megfelelő kiíratási programrészt!

Megoldás. A program:

```
   |  DO 1 I=1,15                                        (6.92)
 1 |  WRITE (5,12) (A(I,J), J=1,15)
12 |  FORMAT(E30.6, 4E18.6/E30.6, 4E18.6/E30.6, 4E18.6/)
```

A 12-es FORMAT utasításban találunk törtvonalat mind a lista közepén, mind a lista végén. A lista közepén elhelyezett törtvonalak a sorváltást biztosítják; itt nincs sorkihagyás (üres rekord). A FORMAT lista végén elhelyezett törtvonal viszont a feladat által megkövetelt üres sor kihagyásáról gondoskodik. A specifikációkat úgy adtuk meg, hogy a számanyag lapközépre kerüljön, és az egyes számok között 5—5 elválasztó betűköz karakter is álljon. A középre állítást és a betűköz nyomtatást a „hagyományos” módon, a mezőszélesség növelésével érjük el.

Az előzőnek teljesen megfelelő szabály mondható ki a beolvasás esetére: *Beolvasási utasításokhoz rendelt* FORMAT *utasításban, a* FORMAT *lista elején vagy végén elhelyezett k számú törtvonal k számú input rekord átugratását (kihagyását), a* FORMAT *lista közepén elhelyezett k számú törtvonal k — 1 számú input rekord átugratását (kihagyását) eredményezi.* Speciálisan a FORMAT lista közepén álló egyetlen törtvonal áttérést jelent a soronkövetkező rekordra.

k) Vezérlő karakterek

Ebben az alpontban néhány olyan lehetőséget ismerünk meg, amelyeket ugyancsak a nyomtatási kép vertikális elrendezésének kialakításában használhatunk fel. Az egész alpont *kizárólag sornyomtató outputra* vonatkozik.

A sornyomtatón keresztül kiadott output rekord 1-es számú (bal szélső) karakterpozíciójának a gép különleges értelmet tulajdonít: nem nyomtatja ki, hanem ún. *vezérlő karakterként* értelmezi.

A legfontosabb és leggyakrabban használt vezérlő karakter az 1-*es számjegy*. Jelentése: *új lap nyitása* a sornyomtatón. Ha tehát az output rekord bal szélső karakterpozícióján egy 1-es számjegy áll, akkor a rekord nem csupán a következő sorban, hanem a következő lap első sorában nyomtatódik. Magát az 1-es számjegyet a gép nem nyomtatja ki.

Azt, hogy az output rekord első karaktere 1-es számjegy legyen, sokféleképpen el lehet érni, de leggyakrabban erre a célra a H specifikációt szokták alkalmazni.

Példa

Egészítsük ki a (6.91) programot olyan lapváltási specifikációval, amelynek hatására a gép a nyomtatási képet új lapon kezdi. Hagyjunk a lap tetején három sor felső margót.

Megoldás. A módosított FORMAT utasítás a következő:

```
4|  |FORMAT(1H1///,51X,18HFUGGVENYTABELLAZAS//////
 | X|51X, 1HX, 12X, 4HF(X)///)
```
(6.93)

Az 1H1 specifikáció a lapváltás jele, ennek segítségével érjük el, hogy a rekord első karakterpozíciójába 1-es kerüljön. Megjegyezzük azonban, hogy ha a K egész típusú változó aktuális értéke 1, akkor lapváltást idézhetünk elő pl. a következő utasítással is:

```
 |  |WRITE (5,4) K
4|  |FORMAT (I1)
```

A következő vezérlő karakter a 0-*s számjegy*. Jelentése: *dupla soremelés*. Hatására a kurrens sornyomtató sor előtt a gép kiír egy üres sort ugyanúgy, mintha a FORMAT specifikáció egy törtvonal karakterrel kezdődne. Használata szintén H specifikációval a legcélszerűbb; ez többnyire az 1H0 specifikáció. Mivel a törtvonal karakter segítségével azonos hatás érhető el, ezért ez a vezérlő karakter nélkülözhető.

A harmadik vezérlő karakter a „+" *jel*. Ennek hatására a rekord kiírásakor a sornyomtató egyáltalában nem emel sort, tehát az *új rekordot az előzőre írja rá*. Elég ritkán, főként speciális nyomtatások esetén (ábrák, grafikonok nyomtatása stb.) alkalmazzák, amikor az output rekordot valamely oknál fogva csak több részletben lehet kiírni. Ilyenkor az első kiíráskor a később kitöltendő karakterpozíciókra betűközöket íratunk, majd a későbbi „újranyomtatások" folyamán fokozatosan kitöltjük ezeket a helyeket.

Konkrét gépi reprezentációkban további specifikus vezérlő karakterek is lehetnek, de ezek ismertetésétől eltekintünk. Hatásuk általában valamiféle „részleges lapváltás". A „fél lapváltás" pl. azt jelenti, hogy ha a sornyomtató már egy fél lapnál többet nyomtatott a kurrens lapra, akkor új lapot nyit, ha pedig még a fél lapon innen van, akkor a lap közepére ugrik. Hasonlóan értendő a harmad és a negyed lapváltás is.

Ne feledkezzünk meg arról, hogy az első karakterpozícióban álló karaktert a gép mindig vezérlő karakterként értelmezi, akármilyen céllal is írattuk azt eredetileg oda. Ha pl. a FORMAT lista első eleme egy I3 specifikáció, és a kiíratandó egész változó aktuális értéke mondjuk 176, akkor a sornyomtató az 1-es számjegyet lapváltásra szóló utasításnak tekinti, és a kinyomtatás eredménye a 76-os szám a következő lap tetején.

Végezetül ismételten emlékeztetjük az olvasót arra, hogy a vezérlő karakterek kizárólag a sornyomtatóra vonatkozólag „vezérlők", egyéb output perifériákon ugyanúgy kiíródnak, mint a többi karakter. Azok a karakterek, amelyeknek nincs meghatározott vezérlési jelentése, a kiírás során eltűnnek, vagyis az első pozíción álló karakter sohasem kerül kinyomtatásra.

l) Decimális normálótényező. A P kiegészítő specifikáció

A P kiegészítő specifikáció segítségével a valós konverziós specifikációk (E, F, G és D) hatását módosíthatjuk. A kiegészítő specifikáció alakja:

$$nP \tag{6.94}$$

ahol n tetszőleges (pozitív, zérus vagy negatív) egész szám. Ez nem önálló specifikáció; mindig egy valós konverziós specifikáció előtt áll. Példák a kiegészítő specifikáció elhelyezésére:

3PF9.4 —2P3E15.8 0PG16.7

A kiegészítő specifikáció megelőzi mind a konverziós specifikációt, mind az esetleges ismétlési tényezőt.

A READ utasításhoz rendelt FORMAT-ban előforduló P kiegészítő specifikáció hatása a következő:

1. ha az input mező *nem tartalmaz* decimális exponenst, akkor a mező értéke 10^n-nel osztva kerül beolvasásra;

2. ha az input mező *tartalmaz* decimális exponenst, akkor semmilyen hatás nem lép fel.

Ha a P kiegészítő specifikációt tartalmazó FORMAT utasítás egy WRITE utasítást vezérel, akkor a normálótényező hatása attól függően más és más, hogy a kivitel milyen konverzióval megy végbe:

1. F konverzióval a kiírandó mennyiség az output rekordon 10^n-nel szorozva jelenik meg;

2. E konverzió esetén az átvitt szám értéke nem változik, csupán a tizedespont tolódik el *n* helyiértékkel jobbra (pozitív *n* esetén), ill. balra (negatív *n* esetén), és a decimális kitevő ennek megfelelően módosul;

3. D konverzió esetén a hatás ugyanaz, mint E konverziónál;

4. G konverzió esetén a normálótényező hatástalan, kivéve, ha a kiírt szám olyan, hogy a kiírás az E konverzióra való áttérést teszi szükségessé (lásd a *d)* alpontot), mely esetben a normáltényező hatása a 2. alatt leírtak szerinti.

Ezek a szabályok összhangban vannak egy, az *a)* alpontban kimondott elvvel: ha egy mennyiséget valamely specifikáció szerint kiíratunk, majd ugyanezen specifikáció szerint később visszaolvastatjuk, ugyanazt a számot kell, hogy visszakapjuk. Ennek megfelelően, ha az F specifikációval kiíratott mennyiségek a kimeneti konverzió közben 10^n-nel szorzódnak, szükségszerű, hogy visszaolvasáskor 10^n-nel osztódjanak. Továbbá, ha a kimeneti konverzió közben a normálótényező nem változtatja meg a decimális exponensű szám értékét, akkor a decimális exponensű számokra beolvasáskor is hatástalan kell hogy legyen. A G specifikáció — sajnos — nem teljesíti a fenti elvet, mert ha a szám olyan, hogy a kinyomtatáskor exponens nélküli szám keletkezik, akkor ennek normálótényező hatása alatt történő visszaolvasásakor a visszakapott szám különbözik a kiírttól.

A P kiegészítő specifikációra még a következő szabály is vonatkozik: a normálótényező hatása nemcsak arra a konverziós specifikációra érvényesül, amely előtt el van helyezve, hanem minden E, F, G vagy D specifikációra, amely a FORMAT listán sorrendben ez után kerül értelmezésre. Amikor a gép értelmezni kezdi a FORMAT listát, zérus normálótényezőt tételez fel mindaddig, amíg egy ettől különbözőt nem talál. Ha egyszer a FORMAT listában előfordult egy zérustól különböző normálótényező, akkor ez marad érvényben mindaddig, amíg a kiértékelés folyamán egy újabb normálótényező (azaz egy újabb P kiegészítő specifikáció) ezt meg nem változtatja. A zérus normálótényezőre a 0P kiegészítő specifikáció segítségével térhetünk vissza.

Példák

1. Legyen az U, V, W és Z változók aktuális értéke rendre: –0.3899752, 0.00039766023, 1903.7785 és 1.205669. Tekintsük a következő kiírási utasítást:

```
      WRITE (5,8) U, V, W, Z                              (6.95)
8     FORMAT (5X, 2PF9.5, 5X, E15.8, 5X, G15.8, 5X, F9.4)
```

Mi a (6.95) utasítások hatása?

Megoldás. (6.95) hatására öt karakter szélességű bal margóra, egymástól 5—5 betűköz karakterrel elválasztva, a következő számok jelennek meg az output rekordon:

▽▽▽▽▽–38.99752▽▽▽▽▽▽39.7660230E-05▽▽▽▽▽▽▽1903.7785▽▽▽▽▽▽▽▽▽▽120.5669

(6.96)

Az U változó kiírásakor a normálótényező az 1. esetnek megfelelően működik: a változó aktuális értékének 10^2-nel való szorzását idézi elő, és így a kiírt szám százszorosa a gépen belüli értéknek. V esetében a kiírás 2. szerint megy végbe: a normálótényező hatására a tizedespont a második decimális jegy utáni karakterpozícióra tolódik, egyúttal a decimális exponens kettővel csökken. (A normálótényező nélkül a kiírási formátum 0.39766023E – 03 lett volna). A harmadik változót, W-t, G specifikációval íratjuk ki; minthogy a kiíratandó mennyiség olyan, hogy nem teszi szükségessé az E konverzióra való áttérést, a normálótényező hatástalan; hatása azonban a negyedik változó, Z, kiírásakor újból érvénybe lép, mégpedig ismét az 1. esetnek megfelelően.

Talán nem árt felhívni a figyelmet egy apróságra, amely esetleg elkerülné az olvasó figyelmét. Arról van szó, hogy az *értékes jegyek száma* szempontjából a normálótényező másként viselkedik az F, és másként az E specifikáció esetén. Az F specifikációnál az történik, hogy a szám előbb megszorzódik tíznek a normálótényező által előírt hatványával, majd az eredmény kiíratódik az adott specifikáció szerint. Itt tehát az értékes jegyek száma aszerint nő vagy csökken, hogy a normálótényező hatványkitevője pozitív vagy negatív-e. (Példánkban a pozitív normálótényező kettővel növelte az értékes jegyek

számát.*) Az E specifikáció esetében a normálótényező működésének mechanizmusa pontosan a következő: 1P normálásra a szám „elfoglalja" a közönséges lebegőpontos nyomtatásnál kihasználatlan, tizedespont előtti helyi értéket, vagyis ezzel eggyel megnöveli az értékes jegyek számát. A normálótényező további növelése nem ad újabb értékes jegyeket, mert amenynyivel több jegy kerül a tizedespont elé, annyival kevesebb áll mögötte. Ha a normálótényező negatív, akkor a mantissza előtt értéktelen zérusok jelennek meg, ezért az ebből eredő jegyveszteség elkerülése céljából a mezőszélességet megfelelően növelnünk kell.

5. A FORMAT és az I/O lista kölcsönhatása. FORMAT lista elhelyzése tömbökben

Az előző pontban megismertük a FORMAT lista különböző lehetséges elemeit és ezek funkcióját. Ebben a pontban olyan szabályokat ismerünk meg, amelyek nem a FORMAT lista egyik vagy másik elemére vonatkoznak, hanem a *lista egészére*. Közelebbről, olyan szabályokról lesz szó, amelyek arra nézve adnak útmutatást, hogyan történik a FORMAT és az I/O listák elemeinek egymáshoz rendelése.

Ilyen szabályokat már az eddigiekben is megismertünk. Láttuk, hogy mind az I/O, mind a FORMAT lista értelmezése balról jobbra halad, és az egymáshoz rendelés lényegében az „egyet innen, egyet onnan" elv alapján megy végbe. Azért csak lényegében, mert mindkét listában lehetnek „többszörös" elemek is. Az I/O lista többszörös elemei a tömbazonosítók (melyek a tömb valamennyi elemének átvitelét előírják — lásd a 2. pontot) és az összetett I/O listaelemek (melyek szintén valamilyen elemcsoport átvitelére adnak utasítást). Minthogy minden átvitelre kerülő változóhoz kell hozzárendelve lennie egy FORMAT listaelemnek, egy többszörös I/O listaelemhez annyi FORMAT listaelemre van szükségünk, amennyi az I/O listaelem szerint átviendő változók száma. Azonban a FORMAT listának is lehetnek többszörös elemei, mégpedig — eddigi isme-

* A külső számábrázolásban a jegyek száma csak a belső ábrázolás adott pontosságának megfelelő mértékben növelhető; ennél nagyobb mértékű növelés esetén a kapott további jegyek már értéktelenek.

reteink szerint — azok az elemek (specifikációk), amelyek előtt ismétlési tényező áll. A kétféle lista együttes feldolgozása során az „egyet innen, egyet onnan" elvet tehát szabatosan úgy kell érteni, hogy mindkét listában minden egyes elemet multiplicitásával (ismétlési számával) veszünk figyelembe.

Ebben a pontban egy olyan lehetőséget ismerünk meg, amely lehetővé teszi, hogy FORMAT listaelemek egy *csoportját* (tehát nemcsak egyetlen FORMAT listaelemet) elláthassunk ismétlési tényezővel, és ily módon FORMAT listaelem-csoportokat képezzünk.

Ilyen összetett FORMAT listaelemhez — vagy, ahogy nevezni szokták, *ismétlési csoporthoz* — jutunk, ha a FORMAT lista tetszőleges részét kerek zárójelek közé zárjuk, és a kezdőzárójel elé egy pozitív egész ismétlési tényezőt írunk.

Például ilyen ismétlési csoportot tartalmaz a következő FORMAT utasítás:

```
| |FORMAT(1H1,56X,8HTABLAZAT,///,15X,5(I10,F8.3))
```
(6.97)

Itt az ismétlési csoport két FORMAT listaelemből, I10-ből és F8.3-ból áll; az ismétlési tényező az 5-ös szám. Egy ismétlési csoport akárhány FORMAT listaelemet tartalmazhat.

Az ismétlési csoport lényegében rövidített írásmód; egyetlen eltéréstől eltekintve (lásd alább) egyenértékű azzal, mintha a csoport elemeit annyiszor egymás mellé írtuk volna, amennyi az ismétlési tényező. A (6.97) példát tekintve, az 5(I10, F8.3) ismétlési csoport egyenértékű a következővel:

I10, F8.3, I10, F8.3, I10, F8.3, I10, F8.3, I10, F8.3

A FORMAT lista értelmezésekor tehát az összetett FORMAT listaelemet annyiszor kell figyelembe venni, amennyi az ismétlési tényező — ugyanúgy, mint ahogy ezt az ismétlési tényezővel ellátott specifikáció kapcsán is láttuk (pl. 5F9.6).

A karakterátvitelre vonatkozó eddigi valamennyi példánk közös vonása volt, hogy az I/O lista és a FORMAT lista elemeinek száma — a multiplicitásokat is figyelembe véve — megegyezett, vagyis az „egy innen, egy onnan" elv alkalmazása esetén a két listának egyszerre értünk a végére. Következő feladatunk annak a tisztázása, hogy mi történik abban az esetben, ha ez nem teljesül, tehát a két lista párhuzamos feldolgozása során valamelyik előbb ér véget, mint a másik?

Egyszerűbb a helyzet, ha az I/O listának érünk előbb a végére. Ebben az esetben *az átvitel véget ér, és a* FORMAT *lista fel nem dolgozott elemei figyelmen kívül maradnak.*

Jóval bonyolultabb eset, ha a FORMAT lista véget ér, az I/O listának pedig még vannak fel nem dolgozott elemei. Erre az esetre a következő szemantikai szabályok az irányadók:

1. Ha a feldolgozás alatt álló FORMAT listában nincs ismétlési csoport, akkor a FORMAT lista végén befejezzük a kurrens rekordot, új rekordot kezdünk, és *elölről kezdjük* a FORMAT lista értelmezését;

2. ha a feldolgozás alatt álló FORMAT listában van legalább egy ismétlési csoport, akkor az eljárás a következő:

2*a)* visszamegyünk a FORMAT listában — a listát lezáró jobb oldali zárójelet nem számítva — az utolsó olyan jobb oldali zárójelig, amely nem egy Hollerith-konstans egyik karaktere;

2*b)* megkeressük az ezen jobb oldali zárójelhez tartozó bal oldali zárójelet. Ez nyilván az utolsó ismétlési csoport kezdőzárójele.

2*c)* Befejezzük a kurrens rekordot, új rekordot kezdünk, és a FORMAT lista értelmezését az említett ismétlési csoport ismétlési tényezőjétől újra kezdjük.

A 2. szabályból nyilvánvaló, hogy a FORMAT listának az utolsó ismétlési csoportot megelőző részére többé nem térünk vissza, tehát e részt mindenképpen csak egyszer „járjuk be”, míg az utolsó ismétlési csoportot és az utána álló részt a szükségtől függően esetleg többször is.

Példák

1. Tekintsük a következő kiírási utasítást:

```
      WRITE (5,8) (X1(I), Y1(I), X2(I), Y2(I), I=1, 20)
8     FORMAT(1H1, //51X, 18HA VEKTOROK ELEMEI:///,
     X36X,2HX1, 13X, 2HY1, 13X, 2HX2, 13X, 2HY2///,
     X1(33X, 2(F7.4, 5X, E13.6, 5X)))                    (6.98)
```

Mielőtt a kiírás részleteibe bocsátkoznánk, vegyük észre, hogy a 8-as címkéjű FORMAT utasításban egy kétszeresen egymásba

skatulyázott ismétlési csoportból álló rendszer található, melyek közül a belsőnek az ismétlési tényezője 2, a külsőé a formális jelentőségű 1 (lásd alább!). Az ismétlési csoportok kétszeres mélységig egymásba skatulyázhatók.

A legtöbb gépi reprezentációban az ismétlési csoportok tetszőleges mélységben is egymásba skatulyázhatók.

Ugyancsak az ismétlési csoportok szintaxisára vonatkozik a következő szabály: *Az ismétlési tényező elhagyható a csoport kezdőzárójele előtt; ebben az esetben* 1*-nek vesszük*. A példabeli FORMAT utasítás külső ismétlési csoportjának tényezőjét tehát akár el is hagyhattuk volna.

A (6.98) utasításpár jelentésére térve, először is megállapíthatjuk, hogy az négy húszelemű vektornak, X1-nek, Y1-nek, X2-nek és Y2-nek a kinyomtatását írja elő. Kérdés azonban, milyen formátum szerint történik a kinyomtatás? A FORMAT lista mindenesetre egy sereg szerkesztési specifikációval kezdődik: új lap nyitása, kétsoros felső margó, egy címszöveg, két üres sor, egy fejléc, és végül két újabb üres sor. Ezután következik a külső ismétlési csoport, amely egy 33 karakter szélességű bal margóval kezdődik, majd eljutunk a belső ismétlési csoporthoz. Ebben vannak a konverziós specifikációk. Az első F7.4 specifikáció szerint kiíródik X1(1), majd öt betűköz karakter, és az E13.6 specifikáció szerint Y1(1) aktuális értéke következik. Ezt újabb öt betűköz karakter követi, amivel elérkeztünk a belső ismétlési csoport végéhez. Ennek ismétlési tényezője 2, ezért most vissza kell térnünk az F7.4 specifikációra. Ennek vezérlete alatt nyomtatódik X2(1), majd — az előzőkhöz hasonlóan — öt betűköz kinyomtatása után az E13.6 specifikáció szerint Y2(1).

Most már elérkeztünk a belső ismétlési csoport végéhez, és egyben a külsőéhez is, hiszen ez utóbbi ismétlési tényezője 1 volt. A FORMAT lista tehát véget ért, azonban még egy sereg elem vár kiírásra. A FORMAT lista értelmezésére vonatkozó 2. szabály szerint ilyenkor vissza kell mennünk a FORMAT listában a „legközelebbi" jobb oldali zárójelig (a 2*a* szabályban pontosan meghatároztuk, hogy ez mit jelent), meg kell keresnünk az ennek megfelelő bal oldali zárójelet, és innen kezdve újra kell értelmeznünk a FORMAT listát. A leírtak

szerint a külső ismétlési csoport kezdetére jutottunk vissza, tehát az X1(2), Y1(2), X2(2) és Y2(2) elemeket rendre F7.4, E13.6, F7.4 és E13.6 specifikációk szerint nyomtatjuk ki. Ezt követőleg újra visszatérünk a külső ismétlési csoport elejére, és így nyomtatjuk ki a 3-as indexű vektorelemeket s í. t. Ez a FORMAT utasítás tehát táblázatosan nyomtatja ki a négy vektor elemeit, az azonos indexűeket egy sorban, és a számokat 5—5 betűköz karakterrel elválasztva.

A (6.98) példán láthatjuk, hogy a cím, a fejléc és a táblázat nyomtatása egyetlen WRITE utasítással elintézhető, míg eddigi ismereteink szerint ehhez legalább kettőre lett volna szükség, és az utóbbit egy 20 menetes DO ciklusba kellett volna beágyazni. A program kidolgozását „hagyományos" programozási eszközök felhasználásával az olvasóra bízzuk.

Az egész példában a sarkalatos fogás az 1-es ismétlési tényezővel ellátott külső ismétlési csoport alkalmazása. Vegyük észre, hogy ebben nem is az ismétlés a lényeg, hiszen valójában ismétlés nem is történik. A csoport rendeltetése annak a helynek a kijelölése, ahonnan kezdve a FORMAT lista értelmezését újra kell kezdeni.

A (6.98) kiírási utasítások végrehajtása nyomán létrejövő nyomtatási kép a következő lehet:

(2 üres sor)

(51 sp. kar.) A VEKTOROK ELEMEI:

(2 üres sor)

	X1	Y1	X2	Y2
	(2 üres sor)			
összesen 20 sor	−3.4283	−0.634834E−03	2.9947	0.766503E 02
	−9.4058	−0.657762E−03	5.4079	0.905442E 02
				
	7.4307	−0.789960E−03	8.5304	0.305771E 03

A FORMAT lista feldolgozására vonatkozó 1. előírás biztosítja a (6.52) programrészlet helyes működését. A program FORMAT utasításában nincs ismétlési csoport, ezért itt a

FORMAT lista végére érve új rekordot kezdünk, és értelmezését előlről kezdjük.

2. Legyen a memóriában egy 30×30-as valós tömbünk. A feladat e tömb elemeinek sorfolytonos kinyomtatása, a következő formátum szerint:

a) Tekintettel arra, hogy a mátrix sorai 30 elemet tartalmaznak, nyilvánvaló, hogy nem helyezhetők el egy sornyomtató sorba, tehát azokat tördelnünk kell. Egy mátrixsort 5 sornyomtató sorra tördelünk. Az egyes elemeket E15.8 specifikációval írassuk ki!

b) Az áttekinthetőség érdekében minden egyes mátrixsor (azaz minden ötödik sornyomtató sor) után egy üres sornyomtató sor hagyandó; ezzel választjuk el egymástól a mátrixsorokat.

c) Hasonlóan minden hatodik mátrixsor után kezdjünk új lapot; ily módon a 30×30-as mátrix kiírását összesen öt lapon helyezzük el!

Mindez nem különösebben nehéz feladat; sokban hasonlít is a (6.92) programmal kapcsolatos feladatra. Érdekessé a következő — utolsó — kikötés teszi:

d) Az egész nyomtatási feladatot *egyetlen* WRITE *és egyetlen* FORMAT utasítás segítségével kell megoldanunk.

Megoldás:

```
      WRITE(5,8)((B(I,J),J = 1,30) I=1,30)
8     FORMAT(6(5(10X,E15.8,5E17.8,/),/),1H1)
```
(6.99)

Ez a FORMAT utasítás is tartalmaz egy kétszeresen egymásba skatulyázott ismétlési csoportot. A belső egyszerű specifikációs lista, amely egyetlen sornyomtató sor formátumát határozza meg: 10 karakternyi bal margó, és hat E15.8 formátummal kinyomtatott szám, amelyek két-két betűköz karakterrel vannak elválasztva. A lista végén egy ferde törtvonal karakter jelzi, hogy a hatodik szám után új sort kell nyitni. A belső ismétlési csoport együtthatója 5, vagyis öt sor kinyomtatása után át lehet térni a külső ismétlési csoport soron következő elemének értelmezésére. Ez ismét egy ferde törtvonal karakter, tehát szintén új sor nyitását írja elő; az előzővel együtt ez egy üres sor kihagyását jelenti. Ily módon jön létre mátrixsoronként egy-egy elválasztó üres sor. Ha már hatszor megtörtént

az üres sor kihagyása, akkor befejeztük a külső ismétlési csoport értelmezését, és áttérhetünk a FORMAT lista soronkövetkező elemének feldolgozására. Ez egy „új lap" specifikáció, hatására a gép új lapot nyit, majd visszatér a külső ismétlési csoport elejére. A nyomtatás az előzőkhöz hasonlóan folytatódik egészen addig, míg csak a WRITE lista elemei el nem fogytak.

Ez a példa önmagában is jól illusztrálja a jelen alpontban részletezett szabályokat, de még ki is egészíthetjük azzal, hogy a nyomtatás előtt is előírjuk új lap nyitását, valamint egy címfelirat elhelyezését. Az ily módon kibővített FORMAT utasítás pl. a következő lehet (a WRITE változatlan, így azt nem ismételjük meg):

```
8   |  |FORMAT(1H1, ///52X, 17HA B MATRIX ELEMEI///,
    | X|6(5(10X, E15.8, 5E17.8, /),/),1H1)              (6.100)
```

Bár a karakterátvitelek eddig megismert módszerei — amint azt a tárgyalt példák is bizonyítják — igen sokféle formaszerkesztési lehetőséget biztosítanak, egy szempontból az általunk megismert FORMAT utasítások kétségkívül kissé merevek. A baj alapvetően az, hogy a FORMAT utasítást, a FORMAT listával együtt, a program írásakor le kell rögzítenünk, és azt a program futása folyamán nem változtathatjuk meg pl. a programbeli változók értékétől függően. Röviden szólva, hiányoljuk, hogy a FORMAT utasításnak nem lehet „értéket adni". (Egy ilyen lehetőséget ugyan megismertünk a 4.*h)* alpontban — lásd pl. (6.79) — ez azonban nagyon korlátozott.)

Az alábbiakban egy módszert ismertetünk, mely bizonyos határok között lehetővé teszi a FORMAT lista „menetközbeni" (azaz a program végrehajtása folyamán történő) megváltoztatását, pontosabban, adatszalagról történő beolvastatását.

Az ASA FORTRAN szintaktikus szabályai megengedik, hogy a (6.1) és (6.3) karakterátviteli utasításokban *f tömbazonosító* lehessen. Ebben az esetben a FORMAT listát — az őt bezáró kezdő- és végzárójellel együtt — abban a tömbben kell elhelyezni, amelynek azonosítója az átviteli utasításban el van helyezve. (Maga a FORMAT szó nem tartozik a listához.) Egyetlen megkötés, hogy az ilyen FORMAT lista nem tartalmazhat H specifikációt.

Ennek a lehetőségnek, mint erre már utaltunk, az a gyakorlati haszna, hogy — a programok adatsorozataival együtt — karakterinformációként beolvastathatjuk az input és output formátumot is.

Példa

Legyenek adva lyukkártyán különböző méretű, nem okvetlenül négyzetes mátrixok. Tudjuk róluk, hogy elemszámuk nem haladja meg a 10 000-et, és hogy mindegyik mátrix oszlopfolytonosan van lyukasztva. Minden lyukkártyán 6 szám van, az egyes számok 12—12 karakterpozíciót foglalnak el (az első az 1—12. pozíciókat, a második a 13—24. pozíciókat s. í. t.). A kártyák utolsó nyolc karakterpozíciója kihasználatlan; ha a mátrix elemeinek száma nem osztható hattal, akkor az utolsó kártyán több kihasználatlan karakterpozíció is lehet. Minden egyes mátrixhoz mellékelünk egy legelső lyukkártyát, amelynek első és második négy karakterpozíciójában a rendszámok (N = a sorok és M = az oszlopok száma) vannak elhelyezve, a 9.—80. karakterpozíciókban pedig a kinyomtatási utasítás FORMAT listája.

A feladat az egyes mátrixoknak a legelső kártyákon adott formátum szerinti sorfolytonos kinyomtatása. Az egyes mátrixok számát nem adjuk meg külön vezérlőkártyán, hanem a kártyaköteg legvégén elhelyezünk egy üres kártyát. Mivel ez a kártya olyan helyen van, ahol a gép „első" (tehát rendszámokat és FORMAT információt tartalmazó) kártyát vár, abból tudjuk meg, hogy vége a bemenő információnak, hogy a beolvasott mindkét rendszám zérussal egyenlő (az üres numerikus mezőt ui. a gép zérusnak veszi). Tehát a program végét az N = 0 és M = 0 feltétel teljesülése jelenti. Feltesszük, hogy a programot olyan gépi reprezentációban írjuk, amelyben az elemszélesség (lásd a 4.*g)* alpontot) 8 karakter. Teljes program írandó, amely a programozó tetszésétől függően egy, vagy több szegmensből állhat.

Megoldás. A feladatra két megoldást adunk, mégpedig egyet, amelyik egy szegmensből, és egy másikat, amely két szegmensből áll. „Kifelé" mindkettő egyformán működik; az egy szegmenses program talán valamivel egyszerűbb. A programok:

```
      MASTER MTXWR
C     ELSO POGRAMVALTOZAT
      DIMENSION X(10000), FORM(9)
    4 READ (2,5) N, M, FORM
      IF(N.EQ.0.AND.M.EQ.0) GO TO 3
      NUMB = N*M
      READ (2,6) (X(I), I=1,NUMB)
      DO 2 I=1,N
      LIM = I+(M-1)*N
    2 WRITE (5,FORM) (X(J), J=I, LIM, N)
      GO TO 4
    3 STOP
    5 FORMAT (2I4, 9A8)
    6 FORMAT (6F12.4)
      END
```

(6.101)

```
      MASTER MTXWR
C     MASODIK PROGRAMVALTOZAT
      DIMENSION X(10000)
      COMMON/FORMVEK/FORM(9)
    4 READ (2,5) N, M, FORM
      IF(N.EQ.0.AND.M.EQ.0) GO TO 3
    5 FORMAT(2I4, 9A8)
      CALL RWRSBR (X, N, M)
      GO TO 4
    3 STOP
      END
```

(6.102)

```
      SUBROUTINE RWRSBR (B, N, M)
      DIMENSION B(N,M)
      COMMON/FORMVEK/FORM(9)
      READ (2,6) B
      DO 2 I=1, N                                   (6.103)
    2 WRITE (5,FORM) (B(I,J), J=1,M)
      RETURN
    6 FORMAT (6F12.4)
      END
```

Legyen az első beolvasandó és kiíratandó mátrix 5×8-as. Ebben az esetben a mátrix bevezető lyukkártyáján a következő állhat:

|▽▽▽5▽▽▽8(22X, E13.6, 4E16.6)

Ennek hatására a gép beolvassa a mátrix 40 elemét és kinyomtatja azt a beolvasott FORMAT lista szerint, azaz úgy, mintha WRITE utasításhoz a következő FORMAT volna hozzárendelve:

FORMAT(22X, E13.6, 4E16.6)

(22 karakter szélességű bal margó, lebegőpontos decimális kinyomtatás, 3—3 elválasztó betűköz karakter.)

Tegyük fel most, hogy a soronkövetkező mátrix már jóval nagyobb méretű: 50×60-as. Ennek bevezető kártyája a következő lehet:

|▽▽50▽▽60(10(16X, E13.6, 5E15.6,/))

Ez a kártya a kinyomtatásra azt írja elő, hogy a 60 számból álló mátrixsorokat 10 sornyomtató sorra kell széttördelni, és minden mátrixsor után egy-egy üres elválasztó sort kell kihagyni. A bevezető kártyán beolvasott FORMAT-információ a kinyomtatási utasításhoz a következő fiktív FORMAT utasítást rendeli:

FORMAT(10(16X, E13.6, 5E15.6,/))

Minthogy az így beolvastatott FORMAT listán nem szerepelhet H specifikáció, ily módon nem szervezhetjük új lap nyitását, valamint címfeliratok nyomtatását. Ezek megoldása a 4. *g)*, *h)* és *k)* alpontokban leírtak szerint végezhető.

A kétféle programváltozat között nincs lényeges elvi különbség. A második megoldás jónéhány programsorral hosszabb, mint az első, mégis van bizonyos elvi érdekessége. A dinamikus indexhatárok lehetőségének (l. Ötödik fejezet 4. pont) kihasználására gondolunk. Mint látható, a főprogramban szereplő X tömb egyindexes, míg a RWRSBR szubrutinban a megfelelő B formális paraméter tömb kétindexes, és méreteit dinamikusan deklaráltuk. Ezzel elkerültük a következési függvény kiszámításának FORTRAN utasításokkal való programozását, ami a (6.102) változatban szükséges volt. Ilyen egyszerű programokban, mint a jelenlegi, ez nem gazdaságos megoldás, azonban bonyolultabb feladatokban az itt bemutatott fogás sokszor hasznos lehet.

Ezzel az ASA FORTRAN nyelv szabályainak ismertetését befejeztük. A következő pontban — most már a teljes anyag birtokában — két teljes FORTRAN programot mutatunk be.

6. Példák teljes FORTRAN programokra

a) Nagyméretű mátrixok szorzása

Gyakorlati számítási feladatok megoldása során igen sokszor előfordul, hogy olyan nagyméretű mátrixokkal kell műveleteket végeznünk, amelyek egészben nem férnek el a gép központi memóriájában. Ilyenkor általában mágnesszalagot alkalmazunk a mátrixok és a részleteredmények tárolására. Úgy is mondhatjuk, hogy *háttérmemóriaként* mágnesszalagot használunk és a mátrixműveleteket „részletekben” hajtjuk végre. Ilyen feladat pl. a következő: legyen adva két, egyenként 200×200-as mátrix, és kiszámítandó ezek szorzata. Tegyük fel, hogy a rendelkezésünkre álló gép memóriakapacitása olyan mértékben korlátozott, hogy a programban mindössze egyetlen 10000 elemű tömböt deklarálhatunk, ennél nagyobb tömb deklarálása pedig egyáltalában nem megengedett.

Bemenő adatok: A program bemenő adatai 6 decimális jegyre megadott valódi tizedestörtek, a rendelkezésre álló bemeneti egység a kártyaolvasó. Az adatlyukasztás módját ezen a szinten már nem írjuk elő; ezt a feladat természetétől függően mindenkor a programozó határozza meg. A formátum előírásakor azt kell szem előtt tartani, hogy a számolóközpontokban általában mind a gépidőért, mind az adatlyukasztásért fizetni kell, mégpedig a felhasznált gépórák, ill. a lyukasztott kártyák számának arányában.

Eredmények kiírása: a program eredményeit a sornyomtatón kell kinyomatni. A program főcíme:

200-AD RENDU NEGYZETES MATRIXOK SZORZASA

Ezt az első lap tetején középen kell elhelyezni. A sornyomtatón nemcsak a szorzatmátrixot, hanem a szorzandókat is ki kell nyomtatni. Ennek megfelelően az eredmény három részből áll; az egyes alcímek rendre a következők:

AZ A MATRIX

A B MATRIX

AZ A∗B MATRIX

A formátum a programozó tetszése szerint választható, figyelembe véve a bemenő mátrixok elemeinek nagyságát és pontosságát. Az előzőkből nem következik, de feltesszük, hogy az **A∗B** mátrix is ugyanolyan tulajdonságú, mint **A** és **B** (azaz elemei 1-nél kisebbek, és 6 tizedesjegyre pontosak). Mivel a program igen nagymennyiségű kimenő adatot állít elő, feltétlenül kikötjük a sornyomtató lapjainak megszámozását; egyébként is külön hangsúlyozzuk az eredménytáblázatok áttekinthetőségének fontosságát.

A feladat végeredményben három részből áll:

1. az input formátum pontos meghatározása;
2. az output formátum pontos meghatározása;
3. a program megírása.

Mellékfeltétel a gazdaságosság mind az adatelőkészítésben, mind a felhasznált gépidőben.

Megoldás

Minthogy olyan programot kell írnunk, amelynek végrehajtása során nem lehet akár az egyik mátrix-komponens is teljes egészében a központi memóriában, a bemenő mátrixokat mágnesszalagra kell vinnünk. A programozási célszerűség érdekében kívánatos **A**-t és **B**-t különböző szalagokra vinni, ellenkező esetben ugyanis a mágnesszalag előre-hátra forgatása áttekinthetetlen programozási bonyodalmakat és rengeteg időveszteséget okozna. Mátrixok szorzásáról, tehát sor—oszlop szerinti műveletről lévén szó, nyilván **A**-t sorfolytonosan, **B**-t pedig oszlopfolytonosan célszerű a mágnesszalagon elhelyeznünk, és ez egyben fontos szempontot ad az adatelőkészítéshez is.

A feladatnak összesen 80 ezer bemenő adata van. Ilyen nagyszámú bemenő adat esetén különösen fontos az input kártyák karakterpozícióinak gazdaságos kihasználása. Éppen ezért ennél a programnál a tizedespont nélküli beolvasási formátum mellett döntünk. Ily módon a számok elhelyezhetők 7 karakterpozícióban (6 decimális jegy + előjel), egy betűköz karaktert pedig elválasztóként hagyhatunk. Így egy 80 karakterpozíciós kártyán tíz számot helyezhetünk el, tehát a teljes számanyagot 8 ezer kártyán tudjuk elhelyezni. (Gondoljuk meg, hogy ha egy számot nem hét, hanem nyolc pozícióban helyeznénk el szintén egy elválasztó betűközzel, akkor egy kártyára már csak 8 szám volna felvihető, és így a teljes számanyag elhelyezése nem 8, hanem 10 ezer kártyát igényelne. Tehát számonként egyetlen karakterpozíció megtakarítása 2000 lyukkártya megtakarítását teszi lehetővé!)

Ezzel teljesen meghatároztuk a bemenő formátumot: tizedespont nélküli (egész szám alakú) lyukkártya-input, egy kártyán 10 szám, egy-egy betűköz karakterrel elválasztva. Az **A** mátrix sorfolytonosan, a **B** mátrix oszlopfolytonosan lyukasztandó.

Áttérünk most az output formátumra. Nyilvánvaló, hogy a kinyomtatás során a mátrixok sorait tördelnünk kell; a 200 számból álló mátrixsorokat (oszlopokat) 20, egyenként 10 számot tartalmazó sornyomtató sorra bontjuk fel. A számanyagot ezúttal nem fogjuk lapközépre állítani, hanem a bal margót a mátrixsorok megszámozására használjuk fel. (A pontos karak-

terbeosztást a 23-as és 26-os címkéjű FORMAT utasításokban adjuk majd meg.)

A mátrixok lyukasztásának, valamint a mágnesszalagra való felvitelnek a formátumából (**A** sorfolytonosan, **B** pedig oszlopfolytonosan kerül lyukasztásra) következik, hogy a kinyomtatáskor is **A** sorfolytonosan, **B** pedig oszlopfolytonosan adódik. Ezt a körülményt a sornyomtató outputon jelezni fogjuk egy emlékeztető felirat formájában.

Mivel a forgalomban levő sornyomtatók (a típustól és a sortávolságtól függően) kb. 70 sort képesek egy lapra nyomtatni, a nyomtatási kép áttekinthetősége érdekében minden harmadik mátrixsor (oszlop) kinyomtatása után új lapot nyitunk. A kinyomtatási utasításokkal kapcsolatban feltesszük, hogy a 2-tes számjegy vezérlő karakterként fél lapváltást, a 3-as pedig harmad lapváltást jelent, vagyis ugrást a sornyomtató lap soron következő felére, ill. harmadára (Hatodik fejezet *4k)* alpont).

Programunk szervezése ezek után a következő. A program első része a kártyáról történő beolvasást, a mágnesszalagra való felvitelt és a mátrixkomponensek kinyomtatását tartalmazza. A beolvasást, felvitelt és kinyomtatást az **A** és **B** mátrixokra vonatkozó egy-egy programciklus végezteti el. Ezután térünk át a tulajdonképpeni mátrixszorzásra. Minthogy egyik mátrixtényezőt sem tarthatjuk egyszerre teljes egészében a központi memóriában, kénytelenek vagyunk az egyik mátrixot négy részletben, a másikat pedig soronként beolvasni. A kényelmes programozás, valamint a mágnesszalagok forgatásának minimumra csökkentése szempontjából legcélszerűbb, ha a **B** mátrixnak egyszerre 50—50 oszlopát olvastatjuk be, és ezeket szorozzuk az **A** mátrix egyenként beolvastatott soraival. Ezáltal az **A**∗**B** mátrix a mátrixszorzás folyamán sorfolytonosan keletkezik.

A programban lényegében három tömböt deklarálunk a számítás adatainak tárolására. (Szerepel egy negyedik azonosító is, ez azonban ekvivalens az elsővel.) A BUFF nevű, tízezer elemű (200×50-es) tömb a **B** mátrix egy-egy negyedrészét, a BUFFA nevű, 200 elemű tömb az **A** mátrix egy-egy sorát, a BUFFC nevű, 200 elemű tömb pedig az eredménymátrix pillanatnyi sorát tárolja. A BUFFRW nevű, 200 elemű tömb ekvivalens BUFF első 200 elemével, és csak a program első részében, a mágnesszalagra vitelkor kerül felhasználásra.

A program feltételezett logikai perifériaszámai a következők:

kártyaolvasó: 2;
sornyomtató: 5;
első mágnesszalag: 7;
második mágnesszalag: 8.

A program:

```
      MASTER MLT 200
      DIMENSION BUFF(200,50), BUFFA(200),
     X              BUFFC(200), BUFFRW(200)
      EQUIVALENCE(BUFF(1), BUFFRW(1))
      J=2
      WRITE(5,20)
      DO 1 I=1,200
      READ(2,21) BUFFRW
      WRITE(7) BUFFRW
      IF(I.EQ.1.OR.MOD(I,3).NE.1) GO TO 2
      WRITE(5,22)J
      J = J+1
    2 WRITE(5,23) I, BUFFRW
    1 WRITE(5,24)
      ENDFILE 7
      REWIND 7
C▽▽▽▽ EZZEL VEGET ERT AZ ELSO MATRIX
C     FELVITELE ES KINYOMTATASA
      J = J+1                                    (6.104)
      WRITE(5,25)
```

```
      DO 4 L=1,4
      READ(2,21) BUFF
      WRITE(8) BUFF
      DO 4 K=1,50
      NC = 50*(L-1)+K
      IF(NC.EQ.1.OR.MOD(NC,3).NE.1) GO TO 9
      WRITE(5,22) J
      J = J+1
    9 WRITE(5,26)NC,(BUFF(INDEX,K),INDEX=1,200)
    4 WRITE(5,24)
      ENDFILE 8
      REWIND 8
      EZZEL VEGET ERT A MASODIK
      MATRIX FELVITELE ES KINYOMTATASA.
      MATRIX SZORZAS KEZDETE.
      J = J+1
      WRITE(5,27)
      DO 5 I=1,200
      READ(7) BUFFA
      DO 7 L=1,4
      READ(8) BUFF
      DO 7 K=1,50
      S=0.0
      DO 8 INDEX=1,200
    8 S = S+BUFFA(INDEX)*BUFF(INDEX,K)
```

```
 7      BUFFC(I)=S
        REWIND 8
        IF(I.EQ.1.OR.MOD(I,3).NE.1) GO TO 6
        WRITE(5,22)J
        J = J+1
 6      WRITE(5,23) I,BUFFC
 5      WRITE(5,24)
        REWIND 7
        WRITE(5,28)
        STOP
20      FORMAT(1H1, 1H2, 40X,
      1 40H200-AD RENDU NEGYZETES MATRIXOK SZORZASA,
      2 /, 1H1, 54X, 11HAZ A MATRIX, /48X,
      3 23H (SORFOLYTONOS KIIRATAS))
21      FORMAT (10F8.6)
22      FORMAT(100X, I4, 5H.  LAP)
23      FORMAT(I4, 5H.  SOR, 5X, 20(F9.6, 9F10.6, /14X))
24      FORMAT(1H3)
25      FORMAT(55X, 10HA B MATRIX, /47X,
      1 26H (OSZLOPFOLYTONOS KIIRATAS))
26      FORMAT(I4,8H. OSZLOP,2X,20(F9.6,9F10.6,/14X))
27      FORMAT(54X, 13HAZ A*B MATRIX, /48X,
      1 23H (SORFOLYTONOS KIIRATAS))
28      FORMAT(37X, 14HPROGRAM VEGE.
      1 29HKOSZONOM AZ OLVASO FIGYELMET.)
        END
```

Úgy hisszük, a program szervezése az előzetes megjegyzések, valamint a közbeiktatott kommentárok alapján jól követhető, ezért a közölt programhoz csupán néhány hozzáfűzni valónk van:

1. Mint a programból látható, az **A** és a **B** mátrix mágnesszalagra vitelét elvégeztető programrészek némileg különböznek egymástól. Az eltérés oka az, hogy míg az **A** mátrixot soronként visszük fel, azaz a 7-es logikai perifériaszámú mágnesszalagon 200 rekordot képezünk, addig **B**-t mindössze 4 rekordra bontjuk fel, tehát egy rekord a 8-as számú szalagon 50 mátrixoszlopnak felel meg. Ezáltal elérjük, hogy a szorzatmátrix egy sorát mindössze 6 mágnesszalaghoz fordulással elő tudjuk állítani: 1 fordulás a 7-es szalaghoz, 4 fordulás a 8-ashoz, és 1 teljes visszacsévélés a 8-as szalagon. (Megjegyezzük, hogy a mágnesszalaghoz fordulások közben a gép bonyolult, ún. „szolgálati" szalagkezelő rutinokat hajt végre, és így nem mindegy, hogy 50 darab 200 szavas rekordot olvastatunk be, vagy egy tízezer szavasat.)

2. Figyelmet érdemelnek a számanyag output formátumát előállító 23-as és 26-os FORMAT utasítások. Ezek működése az előző alpontban ismertetett szabályokon alapul, és az ismétlési csoportok adta lehetőséget használják ki a kívánt output formátum kialakítására. Ez a formátum a következő:

```
141.SOR              -0.664789  -0.808722 ...
                      0.357998   0.622542 ...
20 sornyomtató sor {      :
                     -0.499307   0.542311 ...
```

(Az oszlopok formátuma ugyanilyen, csak a SOR szó helyett OSZLOP lép fel benne.) Az olvasó figyelmét voltaképpen a FORMAT utasításban alkalmazott egyik megoldásra kívánjuk felhívni, amelynek segítségével a „141. SOR" felirat alatt a betűközök kiírását oldottuk meg. Ezt a /14X specifikáció részlet idézi elő, amely egy soremelést és 14 betűköz nyomtatását eredményezi, és ezáltal biztosítja a következő sorokban a sorszám-felirat alatti résznek betűközkarakterekkel való feltöltését.

3. A sornyomtató lapok oldalszámozását úgy oldottuk meg, hogy minden harmadik mátrixsor (oszlop) után — külön

utasítással — kiírattuk a J egész változó értékét, amelyet minden egyes lapkezdés után eggyel megnövelünk. A lapszámot az egyes mátrixok kezdő oldalaira — a sornyomtató sorokkal való takarékosság érdekében — nem íratjuk ki.

4. Mint említettük, a programban önkényes értelmezést tulajdonítunk az 1H2 és 1H3 specifikációknak: bizonyos fajta „vertikális tabulátor"-ként használjuk őket (1H2 fél lapváltás, 1H3 harmad lapváltás). Programunk csak olyan gépi reprezentációban működik helyesen, amelyben ezeknek a vezérlő karaktereknek valóban ilyen jelentése van, és amelyben egy harmad lap legalább 22 — de inkább 23 — sorból áll. Kis ügyeskedéssel a fél és harmad lapváltást programozási úton is elő lehet állítani, de ez feleslegesen növelné az amúgy is eléggé terjedelmes programot.

b) Egy görbeillesztési feladat

Következő feladatunkban egy görbeillesztési (paraméter meghatározási) feladatot oldunk meg. Az ilyen feladatok igen gyakoriak pl. fizikai mérések kiértékelésekor.

A matematikai modell és a feladat leírása: A feladat — általános megfogalmazásban — a következő: legyenek adva valamely $f(x)$ függvény „mért" értékei az $x_1, x_2, \ldots, x_N$ helyeken, és jelölje ezeket $y_1, y_2, \ldots, y_N$. Az $f(x)$ függvényről feltesszük, hogy típusa ismert, de az x független változón kívül még K számú paramétertől is függ; legyenek ezek $a_1, a_2, \ldots, a_K$. A feladat olyan a_i paraméterek meghatározása, amelyek mellett az y_j „mért" értékek a lehető legjobban illeszkednek az $f(x)$ függvényhez. A „lehető legjobb" illeszkedést természetesen igen sokféleképpen lehet definiálni; mi akkor nevezzük legjobban illeszkedőnek az f függvényt, ha a

$$Q = \sum_{j=1}^{N} \frac{1}{\sigma_j^2} (y_j - f_j)^2 \tag{6.105}$$

ún. súlyozott négyzetes eltérés a lehető legkisebb. (6.105)-ben a σ_j^2-k az egyes y_j mért értékek szórásnégyzetei; ezeket az y_j értékekkel együtt kell megadni, vagy előző tapasztalatokból, esetleg elméleti meggondolásokból becsülni. A feladat matematikailag tehát a következő: minimalizálni kell a (6.105) kifejezést az a_i $(i=1, \ldots, K)$ paraméterek szerint. Ezt a felada-

tot, tehát a paraméterek meghatározását a súlyozott legkisebb négyzetek módszere szerint, *görbeillesztésnek* nevezzük. (Némileg egyszerűsíti a feladatot, ha a σ_j^2 mennyiségek mind egyenlők, vagy legalábbis egyenlőknek vehetők, amely esetben súlyozatlan legkisebb négyzetek módszeréről beszélünk.)*

A (6.105)-re kirótt minimumfeltételből következik, hogy Q-nak az a_i-k szerinti parciális deriváltjai zérussal egyenlők. Ez a feltétel egy K ismeretlenes transzcendens egyenletrendszert ad az ismeretlen a_i paraméterekre:

$$q_i = \frac{\partial Q}{\partial a_i} = \sum_{j=1}^{N} \frac{1}{\sigma_j^2}(y_j - f_j)\frac{\partial f_j}{\partial a_i} = 0 \qquad (i=1, \ldots, K) \quad (6.106)$$

(Az egyenleteket −2-vel egyszerűsítettük.)

Ha ismeretes a keresett a_i paramétereknek egy jó kiinduló közelítése, akkor a (6.106) egyenletrendszer megoldható a Newton—Raphson-módszerrel. E módszer lényege — mint a numerikus analízis elemeiből ismert — a következő: tekintsük a

$$\mathbf{q}(\mathbf{a}) = \mathbf{0} \qquad (6.107)$$

K egyenletből álló, K ismeretlenes, transzcendens, vektoriális egyenletrendszert. Jelölje az **a** megoldásvektor l-edik közelítését $\mathbf{a}^{(l)}$; a következő közelítésre az alábbi iterációs formulával térhetünk át:

$$\mathbf{a}^{(l+1)} = \mathbf{a}^{(l)} - \left(\frac{d\mathbf{q}}{d\mathbf{a}}\right)^{-1} \mathbf{q}(\mathbf{a}) \qquad (6.108)$$

ahol $\left(\frac{d\mathbf{q}}{d\mathbf{a}}\right)^{-1}$ a **q** vektor—vektor-függvény **a** szerinti deriváltjának inverzét jelenti. A derivált mátrix elemei a $\frac{\partial q_i}{\partial a_k}$ parciális deriváltak ($i = 1, \ldots, K$ és $k = 1, \ldots, K$).

A (6.108) iterációs formulát (6.106)-ra alkalmazva, és az eredményt koordinátákban kiírva, a következő fő számítási lépésekből álló iterációs eljárásra jutunk:

* A legkisebb négyzetek módszerének kitűnő elméleti és gyakorlati tárgyalását adja pl. *Linnik:* Metod naimenysih kvadratov (Fizmatizdat, 1962) c. könyvében; magyar nyelven l. *Vincze I.:* Matematikai statisztika műszaki alkalmazásokkal (Műszaki Könyvkiadó, 1968).

1. Bemenő adatként kiindulunk az x_j, y_j és σ_j mennyiségekből ($j=1, \ldots, N$), az $\mathbf{a}$ vektor $\mathbf{a}^{(0)}$ kezdeti közelítéséből, valamint egy ε pontossági előírásból.

2. Képezzük az $f_j^{(0)}=f(x_j, a_1^{(0)}, a_2^{(0)}, \ldots, a_K^{(0)})$ mennyiségeket $j=1, \ldots, N$-re.

3. (6.106) alapján képezzük a $q_i^{(0)}$ mennyiségeket $i=1, \ldots, K$-ra.

4. Előállítjuk a $\left(\frac{d\mathbf{q}}{d\mathbf{a}}\right)$ deriváltmátrixot.

5. Előállítjuk a deriváltmátrix inverzét.

6. Alkalmazzuk a (6.108) formulát $l=0$ mellett, és előállítjuk az $\mathbf{a}^{(1)}$ közelítő vektort.

7. Megvizsgáljuk, hogy az $\mathbf{a}^{(1)}-\mathbf{a}^{(0)}$ különbségvektor — valamely később meghatározandó értelemben — elegendően kicsiny-e. Ha nem, visszatérünk a 2. pontra: megismételjük a 2—7. alattiakat $l=1$ mellett és így előállítjuk az $\mathbf{a}^{(2)}$ közelítő vektort.

8. Ha a különbségvektor már elegendően kicsiny, akkor gondoskodunk az eredmények kinyomtatásáról — később pontosabban definiáljuk majd, hogy melyek legyenek ezek — és befejezzük a programot, vagy esetleg felfüggesztjük a végrehajtását, egy újabb adatsorozatra várva.

Mint a (6.106) egyenletek egyszeri differenciálásával könnyen belátható, a (6.108)-ben fellépő deriváltmátrix (i, k)-adik eleme a következő:

$$\frac{\partial q_i}{\partial a_k} = -\sum_{j=1}^{N} \frac{1}{\sigma_j^2} \frac{\partial f_j}{\partial a_k} \frac{\partial f_j}{\partial a_i} + \sum_{j=1}^{N} \frac{1}{\sigma_j^2} \frac{\partial^2 f_j}{\partial a_i \partial a_k} (y_j - f_j) \qquad (6.109)$$

A fenti feladat megoldására teljesen általános programot írni korántsem egyszerű, mivel számos numerikus és egyéb nehézséggel jár. Éppen ezért mi a továbbiakban nem tűzzük ki feladatként az általános program megírását, hanem annak egy erősen leegyszerűsített, de gyakorlati szempontból mégis érdekes változatát tárgyaljuk.

A lehető legegyszerűbb esetben az f függvény lineárisan függ az a_i paraméterektől. Ekkor a (6.106) egyenletrendszer is lineáris, és több-kevesebb numerikus nehézséggel egzaktul, tehát iteráció nélkül is megoldható.

Mi a feladatot itt arra a némileg bonyolultabb speciális esetre oldjuk meg, amikor az f függvényről tudjuk, hogy a következő alakú (vagy legalábbis kielégítő pontossággal ilyennek tekinthető):

$$f(x, a_1, \dots, a_K, \lambda_1, \dots, \lambda_K) = a_1 + \sum_{i=2}^{K} a_i e^{\lambda_i x} \tag{6.110}$$

ahol a meghatározandó paraméterek $a_1, \dots, a_K$ és $\lambda_2, \dots, \lambda_K$. A megoldandó egyenletrendszer tehát $2K-1$ ismeretlenes.

Az f függvény parciális deriváltjai ekkor a következők:

$$\frac{\partial f}{\partial a_i} = e^{\lambda_i x} \qquad (i=1, \dots, K) \tag{6.111}$$

$$\frac{\partial f}{\partial \lambda_i} = a_i x e^{\lambda_i x} \qquad (i=2, \dots, K) \tag{6.112}$$

A (6.111) képlet az $i=1$ esetben is érvényes, ha λ_1-et zérusnak vesszük. Ezek behelyettesítésével a (6.106) egyenletek a következőkbe mennek át:

$$\begin{aligned} q_{2i-2} &= \sum_{j=1}^{N} \frac{1}{\sigma_j^2} (y_j - f_j) a_i \lambda_i e^{\lambda_i x} \qquad (i=2,3, \dots, K) \\ q_{2i-1} &= \sum_{j=1}^{N} \frac{1}{\sigma_j^2} (y_j - f_j) e^{\lambda_i x} \qquad (i=1, 2, \dots, K) \end{aligned} \tag{6.113}$$

A továbbiakban a páros indexű egyenleteket — minthogy azokban a λ_i paraméterek szerinti parciális deriváltak szerepelnek — „λ típusúaknak”, a páratlan indexűeket pedig — az a_i szerinti parciális deriváltak alapján — „a típusúaknak” fogjuk nevezni.

Felírjuk most a (6.113) egyenletrendszerre a (6.109) mátrixelemeket. Ehhez szükségünk van az f függvény másodrendű

parciális deriváltjaira. (6.111) és (6.112) a_k, ill. λ_k szerinti differenciálásával a következőt kapjuk:

$$\frac{\partial^2 f}{\partial a_i \partial a_k} = 0; \quad \frac{\partial^2 f}{\partial \lambda_i \partial a_k} = \frac{\partial^2 f}{\partial a_k \partial \lambda_i} = \begin{cases} 0, & \text{ha} \quad i \neq k \\ x e^{\lambda_i x}, & \text{ha} \quad i = k \end{cases}$$

$$\frac{\partial^2 f}{\partial \lambda_i \partial \lambda_k} = \begin{cases} 0, & \text{ha} \quad i \neq k \\ a_i x^2 e^{\lambda_i x} & \text{ha} \quad i = k \end{cases} \tag{6.114}$$

Ezeket behelyettesítve (6.109)-be, és figyelembe véve, hogy a derivált mátrix páros indexű sorai a „λ típusú" egyenleteknek, páratlan indexű sorai az „a típusú" egyenleteknek felelnek meg; a páratlan indexű oszlopok a λ_i-k szerinti parciális deriváltaknak, a páros indexű oszlopok pedig az a_i-k szerinti parciális deriváltaknak, a mátrixelemekre a következő kifejezéseket kapjuk (a derivált mátrixot **D**-vel jelölve):

$$\mathbf{D}_{2i-2,2k-2} = a_i a_k T_{ik}, \quad \text{ha} \quad i \neq k$$

$$\mathbf{D}_{2i-2,2i-2} = a_i^2 T_{ii} - a_i w_{i-1} \quad (i = 2, 3, \ldots, K; \quad k = 2, 3, \ldots K)$$

$$\mathbf{D}_{2i-2,2k-1} = a_i R_{ik}, \quad \text{ha} \quad i \neq k \tag{6.115}$$

$$\mathbf{D}_{2i-2,2i-1} = a_i R_{ii} - q_{2i}/a_i \quad (i=2, 3, \ldots, K; \; k = 1, 2, \ldots, K)$$

$$\mathbf{D}_{2i-1,2k-2} = a_k R_{ik}, \quad \text{ha} \quad i \neq k$$

$$\mathbf{D}_{2i-1,2i-2} = a_i R_{ii} - q_{2i}/a_i \quad (i = 1, 2, \ldots, K; \quad k = 2, 3, \ldots, K)$$

$$\mathbf{D}_{2i-1,2k-1} = S_{ik} \quad (i, k = 1, 2, \ldots, K)$$

ahol

$$S_{ik} = \sum_{j=1}^{N} e^{(\lambda_i + \lambda_k)x_j} \frac{1}{\sigma_j^2} \tag{6.116}$$

$$R_{ik} = \sum_{j=1}^{N} x_j e^{(\lambda_i + \lambda_k)x_j} \frac{1}{\sigma_j^2} \tag{6.117}$$

$$T_{ik} = \sum_{j=1}^{N} x_j^2 e^{(\lambda_i + \lambda_k)x_j} \frac{1}{\sigma_j^2} \tag{6.118}$$

$$(i, k = 1, \ldots, K)$$

$$w_i = \sum_{j=1}^{N} x_j^2 \frac{y_j - f_j}{\sigma_j^2} e^{\lambda_{i+1} x_j} \quad (i = 1, \ldots, K-1) \tag{6.119}$$

Ezzel teljes egészében meghatároztuk a feladatot; azonban a specializálásban most még egy kicsit tovább megyünk, amennyiben feltesszük, hogy $y_j = \sigma_j^2$, vagyis az egyes bemenő adatok szórásnégyzetei megegyeznek magukkal a bemenő adatokkal.* A bemenő adatok között ily módon nem szerepelnek a σ_j értékek, és ennek megfelelően módosulnak a (6.105), (6.113) és (6.115)—(6.119) formulák is.

Bemenő adatok: A bemenő adatokat három csoportba soroljuk, úgymint:

a) a feladat paraméterei;

b) kezdőértékek;

c) mért értékek.

A feladat paraméterei a következők: N, a mért adatok száma; K, a meghatározandó görbeparaméterek száma; x_0, az első abszcisszaérték; Δx, az abszcissza lépésköze, végül ε, az előírt relatív pontosság, ameddig az iterációban el kell menni. Az előző kettőt maximálisan ötjegyű egész számként, az utóbbiakat valós számként olvastatjuk be; és számukra 15—15 karaktert tartunk fenn az input rekordon.

A kezdőértékeket kettesével olvastatjuk be: egy rekordba kerülnek az összetartozó a_i és λ_i paraméterértékek. Tekintettel arra, hogy λ_1 mindig zérus, kezdőértékét is kötelezően zérusként kell megadni. A kezdőértékeket tartalmazó rekordok 30 karakter hosszúságúak; a beolvasandó kezdőértékek 15—15 karakter szélességű mezőkben vannak elhelyezve.

Végül az y_j „mért" értékeket nyolcasával olvastatjuk be, 10—10 karakter szélességű input mezőkből. Amint az a hasonló feladatokban igen gyakori, feltesszük, hogy ezek az értékek egész számok, vagyis tizedespontot nem tartalmaznak. Számítástechnikai okokból azonban valósként deklaráljuk őket, és így az alkalmas beolvasási specifikáció F10.0.

A beolvasási formátumokat az INOUT nevű MASTER szegmens 20-as, 21-es és 22-es FORMAT utasításaiban találjuk meg.

Eredmények: A program által előállított eredménylapon a program indítása után azonnal kiíratjuk az összes bemenő adatokat, a mért értékek kivételével. (Ezek visszaíratására a számítási eredményekkel együtt kerül sor.)

* Ez a feltevés akkor szokásos, ha az y_j mennyiségek rögzített j mellett Poisson-eloszlást követnek. Fizikai méréskiértékelésekkor ez sok esetben teljesül.

A számítási eredmények a következők: az iterációs lépések száma; az iteráció végeredményeként kapott a_i és λ_i értékek; egy értéktáblázat, amely a mért értékeket, a hozzájuk illesztett számított értékeket, valamint a mért és számított értékek különbségét tartalmazza; a (6.105) formulából számított Q mennyiség (amelyet az eredménylapon „KHI-NEGYZET" felirattal íratunk ki, az ismert statisztikai próba általánosan elfogadott elnevezésének megfelelően); és végül a **D** mátrix inverze, a paraméterek utolsó iteráltjaiból számítva. A kiírási formátumok az INOUT szegmens 24—28 címkéjű FORMAT utasításaiban találhatók meg.

A program működését ellenőriztük egy mintafeladaton, és ennek eredménytáblázatát közöljük (lásd a 2. mellékletet).

A program leírása: A program három szegmensből áll. A főprogram az INOUT azonosítót kapta, és — mint erre neve is utal — kizárólag az I/O műveletek adminisztrálásával foglalkozik. Az érdemi számolás a PROCESS nevű szubrutinban történik, amely részint egy blank közös adatmezőn, részint bizonyos formális paramétereken keresztül tartja a kapcsolatot a főprogrammal. A harmadik eljárás a MATINV nevű mátrixinverzió, amely a **D** mátrixot iterációs lépésenként invertálja. (Ez utóbbi eljárás programját nem írtuk ki részletesen; felhasználható pl. a (3.81) program.)

Mivel a számolás gerincét a PROCESS eljárás képezi, az alábbiakban röviden ismertetjük ennek működését. A (6.113) és (6.115) formulákra rátekintve azonnal láthatjuk, hogy azok kiértékeléséhez igen sokszor kell az exponenciális függvény értékét kiszámítanunk, tehát a program hatékonysága nyilván nagymértékben azon múlik, hányszor fordulunk az EXP standard függvényhez. Ezért a programot úgy állítottuk össze, hogy abban lehetőleg ne kerüljön sor fölösleges függvényhivatkozásokra. Az **S**, **R** és **T** segédmátrixok, valamint a **w** segédvektor bevezetésének programozástechnikai szerepe éppen az, hogy ezek elemeit párhuzamosan számítjuk, így az azonos argumentumú exponenciális függvényekhez csak egyszer kell fordulni.

A PROCESS szubrutin — a fentebb leírtaknak megfelelően — három fő számítási lépést tartalmaz: az első lépésben kerülnek kiszámításra a segédmátrixok és a **w** segédvektor elemei, valamint az f függvény helyettesítési értékei; a második

lépés a **D** mátrix előállítása és invertálása; a harmadik lépés pedig **D** inverzének és a **q** vektornak az összeszorzása, és az újabb iteráltak előállítása. Az egyes iterációs lépések végén konvergenciavizsgálattal ellenőrződik, hogy valamennyi iterált „beállt"-e már ε relatív hibával — vagyis az egymásutáni iteráltak eltérése osztva a megfelelő iteráltakkal, abszolút értékben kisebb-e mint ε —, és a vizsgálat kimenetelétől függően vagy visszaugrás következik egy újabb iterált előállítására, vagy kilépés a szubrutinból. Amennyiben az iteráció 20 lépésben nem konvergál a kívánt pontossággal, akkor egy figyelmeztető szöveg kiírása után kilépés következik a PROCESS szubrutinból.

A programozás és a numerikus eljárás néhány részletkérdésére még visszatérünk. Most álljon itt a teljes FORTRAN program (az okvetlenül szükséges átjelöléseken túlmenően a programban **w**-t Q-val, **D**-t pedig DF-fel jelöljük):

(6.120)

```
      MASTER INOUT
      REAL LAMBDA (5), KHI2
      DIMENSION Y(256), F(256), G(256), R(5,5),
     X S(5,5), T(5,5), FE(9), A(5), Q(4), DF(81)
      COMMON X0, DELTAX, KHI2, NITER, Y, F, G, A,
     X LAMBDA, FE, Q, N, EPS
C     ADATBEOLVASASOK KEZDETE
    1 READ(1,20) N, K, X0, DELTAX, EPS
      L = 2*K-1
      READ (1,21) (A(I), LAMBDA(I),  I=1,K)
      READ (1,22) (Y(I),  I=1,N)
C     BEMENO ADATOK VISSZAIRATASA
      WRITE (2,23) N, K, X0, DELTAX, EPS
      WRITE (2,24) (A(I),  I=1, K)
      WRITE (2,25) (LAMBDA(I),  I=2, K)
```

```
      CALL PROCESS (R, S, T, DF, K, L)
C     EREDMENYEK KINYOMTATASA
      WRITE (2,26) NITER
      WRITE (2,24) (A(I),   I=1, K)
      WRITE (2,25) (LAMBDA(I),   I=2, K)
      WRITE (2,27) (I, Y(I), F(I), G(I),   I=1, N)
      WRITE (2,28) KHI2
      WRITE (2,29)
      DO 2 I=1, L
      LIM = L*(L-1)+I
    2 WRITE (2,30) (DF(KIND),   KIND=I, LIM, L)
      PAUSE 01
      GO TO 1
   20 FORMAT (2I5, 3E15.8)
   21 FORMAT (2E15.8)
   22 FORMAT (8F10.0)
   23 FORMAT (1H1, //44X,
     X33HEXPONENCIALIS FUGGVENY ILLESZTESE//,
     X5X, 17HBEMENO ADATOK: N=, I3, 5X, 2HK=, I2,
     X5X, 2HX=, E13.6, 5X, 7HDELTAX=, E13.6,
     X5X, 4HEPS=, E13.6)
   24 FORMAT(20X, 2HA=, E13.6, 4(4H,▽▽▽, E13.6))
   25 FORMAT (20X, 7HLAMBDA=, 12X, E13.6,
     X                                 3(4H,▽▽▽, E13.6))
   26 FORMAT (/5X, 14HVEGEREDMENYEK:, 1X,
```

```
     X 27HAZ ITERACIOS LEPESEK SZAMA=, I3)
27     FORMAT (/39X, 20HA SZAMITOTT ES MERT▽,
     X 24HERTEKEK OSSZEHASONLITASA,//,
     X 40X, 1HJ, 8X, 4HY(J), 9X, 4HF(J), 7X,
     X 9HY(J)-F(J),//, 1(38X, I3, F13.0, F14.2, F13.2))
28     FORMAT (/5X, 12HKHI-NEGYZET=, E11.4)
29     FORMAT (/5X, 21HA KOVARIANCIA MATRIX:/)
30     FORMAT (1H0, 5X, 9E13.4)
       END
       SUBROUTINE PROCESS (R, S, T, DF, K, L)
       REAL LAMBDA(5), KHI2
       LOGICAL RLERR
       DIMENSION R(K,K), S(K,K), T(K,K), DF(L,L), Y(256),
     X F(256), G(256), FE(9), A(5), Q(4)
       COMMON X0, DELTAX, KHI2, NITER, Y, F, G,
     X A, LAMBDA, FE, Q, N, EPS
       NITER=1
12     RLERR=.TRUE.
       X=X0
       DO 1 JIND=1, N
       F(JIND)=A(1)
       DO 2 LIND=2, K
2      F(JIND) = F(JIND)+A(LIND)*
     X                          EXP(LAMBDA(LIND)*X)
1      X = X+DELTAX
```

```
C       KISZAMITOTTUK A KOZELITO FUGGVENY ERTE
C       KEKET AZ ADOTT PARAMETER ERTEKEK MELLETT
C       KOVETKEZIK S, R, T ES W ELOALLITASA
        DO 3 IIND = 1,K
        DO 3 KIND = IIND, K
        S(IIND,KIND) = 0
        R(IIND,KIND) = 0
        T(IIND,KIND) = 0
        X = X0
        IF(IIND.NE.1) GO TO 5
        K2 = 2*KIND—1
        K2M1 = K2—1
        FE(K2) = 0
        IF(KIND.EQ.1) GO TO 5
        FE(K2M1) = 0
        Q(KIND—1) = 0
      5 DO 7 JIND = 1, N
        WKSP0 = EXP((LAMBDA(IIND)+
       X                    LAMBDA(KIND))*X)/Y(JIND)
        WKSP1 = Y(JIND)—F(JIND)
        S(IIND,KIND) = S(IIND,KIND)+WKSP0
        IF(IIND.EQ.1) FE(K2) = FE(K2)+WKSP0*WKSP1
        WKSP0 = X*WKSP0
        R(IIND,KIND) = R(IIND,KIND)+WKSP0
        IF(IIND.EQ.1.AND.KIND.GE.2) FE(K2M1) =
```

```
     X XFE(K2M1)+WKSP0*WKSP1
       WKSP0 = WKSP0*X
       T(IIN D,KIND) = T(IIND,KIND)+WKSP0
       IF(IIND.EQ.1.AND.KIND.GE.2) Q(KIND—1) =
     X Q(KIND—1)+WKSP0*WKSP1
 7     X = X+DELTAX
       IF(IIND.EQ.KIND) GO TO 3
       S(KIND,IIND) = S(IIND,KIND)
       R(KIND,IIND) = R(IIND,KIND)
       T(KIND,IIND) = T(IIND,KIND)
 3     CONTINUE
       KOVETKEZIK A D MATRIX ELOALLITASA
       DO 6 IIND = 1,K
       I2 = 2*IIND—1
       I2M1 = I2—1
       DO 6 KIND = 1, K
       K2 = 2*KIND—1
       K2M1 = K2—1
       DF(I2,K2) = S(IIND,KIND)
       IF(IIND.NE.1)DF(I2M1,K2) = A(IIND)*
     X                                  R(IIND,KIND)
       IF(KIND.NE.1)DF(I2,K2M1) = A(KIND)*
     X                                  R(IIND,KIND)
       IF(IIND.NE.1.AND.KIND.NE.1) DF(I2M1,K2M1) =
     X A(IIND)*A(KIND)*T(IIND,KIND)
       IF(IIND.NE.KIND.OR.IIND.EQ.1) GO TO 6
```

```
      DF(I2M1,K2) = DF(I2M1,K2)—FE(I2M1)
      DF(I2,K2M1) = DF(I2,K2M1)—FE(I2M1)
      DF(I2M1,K2M1) = DF(I2M1,K2M1)—A(IIND)*
     X                                        Q(IIND—1)
    6 CONTINUE
      DO 8 IIND = 2,K
      I2 = 2*IIND—2
    8 FE(I2) = FE(I2)*A(IIND)
   15 CALL MATINV (DF,L)
C     KÖVETKEZIK AZ INVERZ MATRIX ES AZ
C     EGYENLETEK VEKTORANAK OSSZESZORZASA
      DO 19 IIND = 1, K
      I2 = 2*IIND—1
      I2M1 = I2—1
      WKSP0 = LAMBDA(IIND)
      WKSP1 = A(IIND)
      DO 9 KIND = 1, L
      IF(IIND.EQ.1) GO TO 10
      LAMBDA(IIND) = LAMBDA(IIND)+
     X                        DF(I2M1,KIND)*FE(KIND)
   10 A(IIND) = A(IIND)+DF(I2,KIND)*FE(KIND)
    9 CONTINUE
C     KOVETKEZIK A KONVERGENCIAVIZSGALAT
      IF(IIND.EQ.1) GO TO 26
      IF(ABS((LAMBDA(IIND)—
```

```
     X WKSP0)/LAMBDA(IIND)).GT.EPS)RLERR=.FALSE.
 26    IF(ABS((A(IIND)—WKSP1)/A(IIND)).GT.EPS)
     X RLERR = .FALSE.
 19    CONTINUE
       IF(RLERR) GO TO 11
       IF(NITER.GE.20) GO TO 13
       NITER = NITER+1
       GO TO 12
C      VISSZATERES EGY UJABB ITERACIOS LEPESRE
 13    WRITE (2,20) NITER
 11    KHI2 = 0
       DO 14 JIND = 1, N
       G(JIND) = Y(JIND)—F(JIND)
 14    KHI2 = KHI2+G(JIND)**2/Y(JIND)
       RETURN
 20    FORMAT(/5X, 11HAZ ITERACIO, I3, 1X,
     X 23HLEPESBEN NEM KONVERGALT)
       END

       SUBROUTINE MATINV (A, N)
       DIMENSION A(N,N)
C      IDE HELYETTESITHETO PL  A 3.81 ELJARAS
       ........
       END
```

Úgy érezzük, hogy az előzetes magyarázatok és a közbeiktatott kommentárok alapján a program működése jól követhető, és így csupán néhány megjegyzésre szorítkozunk.

Nem hallgathatjuk el, hogy bár a bemutatott mintafelada-

ton (2. melléklet) minden rendben működött, az alkalmazott numerikus módszernek vannak gyenge pontjai. Mint ismeretes, a többismeretlenes transzcendens egyenletrendszerekre alkalmazott Newton—Raphson eljárás, amelyen a jelen program alapul, csak elegendően jó kezdőértékekből kiindulva konvergál; ellenkező esetben az eljárás divergálhat, vagy egy nem kívánt megoldáshoz is konvergálhat. Éppen ezért, ha ezt a programot valóban használni akarnánk tömeges adatfeldolgozásra, akkor ki kellene egészítenünk egy kezdőérték előállító vagy javító szegmenssel, amely a bemenő adatként megadott kezdőértékeket a keresett megoldás közelébe viszi, és a Newton—Raphson módszer esetleges divergenciája esetén is intézkedik (pl. újabb kezdőértékekkel újra kezdi az iterációt). Ilyen numerikus eljárások ismeretesek az irodalomban, és általában a KHI2 („khi-négyzet") változó direkt numerikus minimalizálásán alapulnak.

Másik, ugyancsak numerikus természetű megjegyzésünk, hogy a (3.81) programon alapuló mátrixinvertáló eljárás sem ideális, mert nincs védve különböző — egyáltalán nem ritka — számítástechnikai nehézségek ellen, mint pl. a nullával való osztás, vagy az igen nagyarányú jegyveszteség. Mi a jelen példában nem törekedhettünk arra, hogy a görbeillesztésnek bármilyen szempontból is optimális módszerét adjuk meg, célunk csupán egy konkrét feladat megoldásának illusztrálása volt.

Ami az alkalmazott eljárás kódolását illeti, egy dologra talán nem árt rámutatnunk, ez pedig a PROCESS eljárásban a formális paraméterek és a közös adatmezőbeli változók megválasztásának elve. Mint a programból látható, formális paraméternek a változó méretű mátrixokat és ezek indexhatárait választottuk, minden egyebet közös adatmezőbe tettünk. A formális paraméterek ilyenfajta felhasználására egyszerű példaként korábban bemutattuk a (6.103) programot, azzal a megjegyzéssel, hogy ott ennek előnyei még nem domborodnak ki eléggé. A (6.120) program viszont már egyértelműen mutatja, hogy a dinamikus indexhatárok alkalmazása messzemenően egyszerűsítheti a kódolást. Az egyméretű tömböket (vektorokat) nem deklaráljuk dinamikusan, mert ez most semmiféle előnyt nem jelentene.

A bemutatott programban a paraméterek megengedett maximális értékei a következők: K = 5, N = 256. Ezek önkényes

korlátok, és a program könnyen átalakítható úgy, hogy ennél tágabb határok között is működjék. Ehhez az INOUT és a PROCESS szegmensek deklarációs részét, valamint az INOUT szegmens egyes kiírási formátumait kell megváltoztatni. Mindenesetre felhívjuk a figyelmet arra, hogy a paraméterek számának növelésével a számítási idő egyre rohamosabban növekszik!

7. Gyakorlatok

1. Írjunk FORTRAN programot egy 200×200-as mátrix transzponálására! (Kisebb méretek esetén a feladatot a (3.66) programrészlettel már egyszer megoldottuk.) Feltesszük, hogy a transzponálandó mátrix az 5-ös logikai perifériaszámú, bináris formátumú mágnesszalagon van elhelyezve oszlopfolytonos elrendezésben, 200 mátrixelemet tartalmazó rekordokban (1 oszlop=1 rekord). Olyan program írandó, amelyben legfeljebb egy olyan tömb van deklarálva, amelynek elemszáma eléri a kétezret, de deklarálhatunk még egy 200 elemű vektort, amely egy mátrixsor vagy -oszlop tárolására elégséges. A transzponált mátrix a 6-os logikai perifériaszámú mágnesszalagon helyezendő el, az 5-ös mágnesszalaggal megegyező elrendezésben.

2. Oldjuk meg a 211. oldal 3. feladatát (lásd (6.32)), de most nem kötjük ki, hogy a javító file rekordjai növekvő sorrendben kövessék egymást! A file-ok a 15. ábrán bemutatott szerkezetűek.

3. Tekintsük a (6.31) programot! Ha ennek 2-tes címkéjű utasítását egy DO ciklusban oldjuk fel:

```
      DO 2 K=1,250
      J = 251-K
2     WRITE (6) REK(J)
```

akkor egy ASA FORTRAN szerint szintaktikusan helyes programrészletet kapunk. Kérdés, hogy szemantikusan egyenértékű-e ez (6.31)-gyel? Ha nem, miben különböznek?

4. Írjunk FORTRAN programot a Pascal-háromszög első 20 sorának előállítására és kinyomtatására! Mint ismeretes, a Pascal-háromszög a binomiális együtthatókból alkotott háromszögmátrix, melynek i, k indexű eleme

$$a_{ik} = \binom{i-1}{k-1} \qquad (i=1, 2, \ldots; \quad k=1, 2, \ldots, i)$$

A binomiális együtthatókra a következő rekurziós formulák érvényesek:

$$a_{ik} = \binom{i-1}{k-1} = a_{i-1,k-1} + a_{i-1,k} = \binom{i-2}{k-2} + \binom{i-2}{k-1} \qquad (6.121)$$

$$(i=3, 4, \ldots; \quad k=2, 3, \ldots, i-1)$$

Definíciószerűen minden i-re $a_{i1} = a_{ii} = 1$. A program az alábbi formátumban nyomtassa az eredménytáblázatot:

```
1

1 1

1 2 1

1 3 3 1

...
```

A táblázat a (6.121) formula jelentését is szemlélteti: a mátrix minden eleme a közvetlenül fölötte álló elemnek és az ettől eggyel balra álló elemnek az összege, a „szélső" elemek pedig mind 1-gyel egyenlők. A táblázat fölött — új lapot nyitva és sorközépre állítva — helyezzük el a következő címfeliratot: A PASCAL HAROMSZOG ELEMEI.

A programban nem szükséges egy 20×20-as tömböt deklarálni; elég egy 20 elemű vektor is, bár ebben az esetben a programozás kétségkívül valamivel több ügyeskedést igényel.

5. Tekintsük a 240. oldalon szereplő példát! Mennyiben módosul a program jelentése, ha a (6.69) programrészletbe a (6.70) FORMAT utasítást helyettesítjük?

6. Egészítsük ki teljes FORTRAN programmá az Ötödik fejezet 6. feladatát (kisrendszámú szimmetrikus mátrix összes sajátértékének meghatározása)! Bemenő adatok legyenek a következők: a mátrix rendszáma (N), a mátrix A_{ik} elemei, valamint az output formátumot meghatározó FORMAT lista. Az eredménylapon szerepeljen az A mátrix, a sajátvektorokból mint oszlopvektorokból összeállított $N \times N$-es mátrix, és a sajátértékek N elemű oszlopvektora. Az eredménylap felirata legyen a következő: SAJATERTEK MEGHATAROZAS PROGRAMJANAK KIPROBALASA. Minthogy az output formátum nyilván függ N-től (például amiatt, hogy az egy sorba nyomtatandó számok száma függ N-től), alkalmazzunk adatszalagról beolvasott FORMAT listát! A szereplő tömbdeklarációk, valamint a FORMAT-adatszalagok megszerkesztésekor feltehetjük, hogy $N \leq 5$.

7. Egészítsük ki teljes FORTRAN programmá az Ötödik fejezet 7. feladatát (magasabbfokú algebrai egyenlet összes gyökének meghatározása Horner-módszerrel). A program bemenő adatai a következők legyenek: az együtthatók vektora (A); az egyes gyökök kezdeti közelítéseit tartalmazó vektor (X0); az A vektor elemeinek a száma (N); végül a pontossági paraméter (EPS). Az eredménylapon szerepeljen az együtthatóvektor, a kezdetiérték vektor, és a gyökök vektora. Tartalmazzon a program „vészkijáratot" arra az esetre, ha valamelyik kezdőértéket rosszul

adtuk meg, és emiatt a HORNER szubrutin válaszparamétere FALSE. Ilyen esetben — hacsak a szubrutin nem az első gyöknél akad el — az addigi számolás folyamán már keletkezett olyan használható részeredmény, amelyet célszerű megmenteni. Ezért kinyomtatandók a már meghatározott gyökök, valamint az ezen gyökök gyöktényezőivel leosztott polinom együtthatói. Kerüljön az eredménylapra egy figyelmeztető felirat, hogy a gyökmeghatározás részben vagy teljesen eredménytelen volt.

8. Egy vállalatnál felmérést végeznek a dolgozók bérhelyzetéről. A vállalat dolgozóinak fizetését kulcsszámok alapján állapították meg. Minden egyes kulcsszámhoz tartozik egy alsó és egy felső bérhatár, és a dolgozó fizetésének ezen bérhatárok közé — esetleg magukra a határokra — kell esnie. (A dolgozó kulcsszámát több paraméter — iskolai végzettség, beosztás, szolgálati idő stb. — alapján állapítják meg, és ezt adottnak vesszük.) A felmérés adataiból ki akarják számítani:

a) az egyes kulcsszámok bérátlagát;
b) az egész vállalati bérátlagot;
c) az egyes kulcsszámok alsó határán levő dolgozók számát;
d) az egyes kulcsszámok felső határán levő dolgozók számát;
e) az egyes kulcsszámokhoz tartozó dolgozók számát.

A feldolgozó programnak ki kell nyomtatnia a vállalati bérlistát, minden egyes dolgozóról megvizsgálva, hogy bére valóban megfelel-e kulcsszáma bérhatárainak; ha nem, akkor hibajelzést kell adnia. (Ez az adatelőkészítés helyességének részleges ellenőrzése.) Ugyancsak adjon hibajelzést a feldolgozó program, ha nemlétező kulcsszámot talál a bérlistán.

A program bemenő adatai a következők:

a) a bérkategóriák kulcsszámainak száma;
b) a vállalati dolgozók száma;
c) az egyes kulcsszámok (legfeljebb ötjegyű egész számok);
d) a kulcsszámokhoz tartozó alsó és felső bérhatárok (legfeljebb négyjegyű egész számok);
e) a vállalati dolgozók nevei (legfeljebb 24 karakterből álló alfanumerikus jelsorozatok);
f) a dolgozó kulcsszáma;
g) a dolgozó alapfizetése.

Az input formátumot közelebbről nem határozzuk meg, de célszerű úgy szervezni, hogy először beolvastatjuk a kulcsszámok és dolgozók számára vonatkozó információt (*a)* és *b)* pont); ezt követőleg a kulcsszámokra vonatkozó információt (*c)* és *d)* pont); végül a dolgozókra vonatkozó információt (*e)*, *f)* és *g)* pont). Az egy kulcsszámra, ill. egy dolgozóra vonatkozó adatok célszerűen egy-egy kártyán (illetve egy lyukszalag rekordban) helyezhetők el.

A program eredménylapja adjon jól áttekinthető táblázatos formában választ a felmérés kérdéseire! Ahol adatelőkészítési hiba gyanúja merül fel, informáljon a hiba jellegéről. A hibásnak talált adatrekordokat a feldolgozás során hagyja figyelmen kívül!

Nem követeljük meg, hogy a bérlista kinyomtatása alfabetikus sorrendben történjék. (Az alfabetikus sorrendbe rendezés ugyanis karaktermanipulációt végző rutinokat igényel, az ASA FORTRAN-ban pedig ilyenek nem állnak rendelkezésre.) A bérlistának az egy dolgozóra vonatkozó adatokat (név, kulcsszám, alapbér) kell tartalmaznia.

A) Függelék

A TELJES ÉS A SZŰKEBB FORTRAN NYELV ELTÉRÉSEI

Említettük az Első fejezet 2. pontjában, hogy a FORTRAN nyelvnek két változata van, egy bővebb (a könyvben mindvégig ezt neveztük ASA FORTRAN-nak és ezt ismertettük), és egy szűkebb, az ún. Basic FORTRAN. A szűkebb nyelv részhalmaza a teljes ASA FORTRAN-nak, tehát a két nyelv eltérései megfogalmazhatók, mint az ASA FORTRAN megszorításai. Az alábbiakban röviden összefoglaljuk ezeket a megszorításokat. Az eltérések összeállításában a könyv felépítése szerint haladunk, és azokat a fő fejezetcímek szerint csoportosítjuk.

1. Aritmetikai és logikai kifejezések. Relációk

A teljes FORTRAN nyelv ötféle értéktípusával (INTEGER, REAL, DOUBLE, COMPLEX és LOGICAL) szemben a szűkebb FORTRAN-ban csak kétféle mennyiség létezik: egész és valós. Ennek megfelelően a számkonstansok is csak valós vagy egész típusúak lehetnek. Valós konstansoknak minden esetben kell tartalmazniuk tizedespontot, akkor is, ha decimális exponenst is tartalmaznak. Az 5E—5 forma nem megengedett. Az aritmetikai műveletek komponenseinek egyező típusúaknak kell lenniük, ez alól az egyetlen kivétel valós alapnak egész kitevőre hatványozása, ami megengedett. Természetesen megengedett valós alapnak valós kitevőre hatványozása is.

A szűkebb FORTRAN-ból hiányoznak a logikai változók, a logikai műveletek, a relációk és a logikai kifejezések. Hiányoznak a logikai konstansok (.TRUE. és .FALSE.) is.

A 4-es táblázat belső standard függvényei közül a szűkebb FORTRAN-ban a következők állnak rendelkezésre: ABS,

IABS, FLOAT, IFIX, SIGN és ISIGN. Ezek definíciója megegyezik a 4. táblázat szerinti definícióval.

Az 5-ös táblázat külső standard függvényei közül azok maradnak meg, amelyek argumentuma is, és függvényértéke is valós típusú (EXP, ALOG, SIN, COS, TANH, SQRT és ATAN), de nem maradnak meg az ALOG 10 és az ATAN2 függvények.

2. Végrehajtható utasítások

Logikai kifejezések hiányában a szűkebb FORTRAN nem ismeri a logikai értékadó utasítást, valamint a logikai IF utasítást sem; így feltétel szerint elágaztatás csak valamely aritmetikai kifejezés előjele alapján lehetséges (aritmetikai IF utasítás).

Nincsenek definiálva a kijelölt GO TO utasítások, és ennek megfelelően hiányzik az ASSIGN utasítás is. A „többirányú elágaztatás" másik lehetősége, a kiszámított GO TO utasítás, a szűkebb FORTRAN-ban is fennáll.

A ciklusutasításokra vonatkozó szabályokban annyi eltérés van, hogy a kiterjesztett hatáskör fogalma nincs definiálva: a ciklus hatáskörén kívülről tilos a ciklusba való bármiféle vezérlésátadás.

3. Deklaratív utasítások. Több szegmensből álló programok

Típusdeklaráló utasítások nincsenek, a változók típusa minden esetben az azonosító első betűje szerint határozódik meg. (I, J, K, L, M és N esetén egész, egyébként valós.)

Tömbméretet deklarálni csak DIMENSION utasításban lehet (COMMON-ban nem). A tömbök csak 1, vagy 2 méretűek lehetnek. A szűkebb FORTRAN nem ismeri a dinamikus indexhatárok fogalmát.

A COMMON utasításokban nem szerepelhetnek blokknevek (azaz csak blank COMMON létezik). A COMMON utasítás listája csupa azonosítóból áll, tömbdeklarátorokat nem tartalmazhat.

A változók kezdőértékének előzetes beállítására a szűkebb FORTRAN-ban nincs lehetőség; a DATA utasítások hiányoznak. Hiányoznak természetesen a BLOCK DATA szegmensek is.

A deklaratív utasításoknak kötött sorrendjük van: első a DIMENSION, ezt követi a COMMON, végül az EQUIVALENCE utasítások következnek.

SUBROUTINE és FUNCTION szegmens paraméterei között nem lehetnek szegmensazonosítók; nem lehet pl. egy SUBROUTINE szegmensnek függvényparamétere. Következésképpen a szűkebb FORTRAN-ban az EXTERNAL utasítás nincs definiálva.

A változókhoz hasonlóan, a függvényérték típusát a függvény azonosítójának első betűje határozza meg. Ennélfogva a FUNCTION alapszó előtt nem helyezhetünk el típusdeklaráló alapszót (pl. INTEGER). Mellékhatások elkerülése céljából a szűkebb FORTRAN nem engedi meg, hogy egy FUNCTION szegmens megváltoztassa akár valamelyik formális paraméterének, akár valamelyik COMMON változónak az értékét.

SUBROUTINE és FUNCTION szegmensek tömbparamétereihez aktuális paraméterként nem adható meg indexes változó, tehát az aktuális paraméter tömb „kezdetének" az Ötödik fejezet 3*a*) alpontja szerinti kijelölésére (lásd az (5.31) programot) nincs lehetőség. Ha a formális paraméter tömb, akkor a megfelelő aktuális paraméternek is tömbnek kell lennie.

4. Adatátviteli utasítások

A szűkebb FORTRAN-ban a FORMAT utasításokban az alábbi specifikációk megengedettek: I, F, E, H, X és „/". (Hiányoznak tehát a következők: G, D, L, A.) Ugyancsak hiányzik a P kiegészítő specifikáció.

Az ismétlési csoportok a szűkebb nyelvben is definiálva vannak, de nem szabad őket egymásba skatulyázni.

A teljes FORTRAN-tól eltérően, az E és F specifikációk hatása beolvasáskor is eltér: ha az input mező tartalmaz decimális exponenst, akkor ezt csakis E specifikációval szabad beolvastatni, F-fel nem. Az F specifikáció kizárólag exponens

nélküli input mezők beolvastatására van fenntartva. (Az exponens jelenléte egyébként az E specifikáció esetén nem kötelező.)

A FORMAT listának tömbökben való elhelyezését a szűkebb FORTRAN nem ismeri, úgyhogy a (6.1) és (6.3)-ban szereplő *f* csak előjeltelen egész szám lehet, tömbazonosító nem.

A sornyomtató vezérlő karakterei a szűkebb nyelvben nincsenek értelmezve. A kinyomtatott rekord bal szélső karakterpozíciójára küldött karakter minden esetben hatástalan, helyette betűköz karakter nyomtatódik.

AZ IBM FORTRAN IV PROGRAMOZÁSI NYELV

Első fejezet: Az ASA és az IBM FORTRAN programozási nyelv

1. Bevezetés

E függelék célja az, hogy az IBM FORTRAN IV nyelvnek a szabványtól való eltéréseit ismertesse.

Magyarországon egyre inkább elterjednek az ESZR gépek, amelyeket ESZR és IBM operációs rendszerekkel üzemeltetnek. Az operációs rendszerekhez hozzá tartoznak a különböző nyelvek fordítóprogramjai, így a FORTRAN fordítók is.

Mint ahogyan azt a könyvből megtanultuk, egy nyelvhez tartozó fordítóprogram a nyelv egy reprezentációját valósítja meg. Az ESZR gépeken működő operációs rendszerek több FORTRAN fordítóprogramot tartalmaznak. Hazánkban a legelterjedtebb a G szintű fordító, de ismertek még a H szintű és az EXTENDED H fordítók is. Ennek megfelelően, az IBM FORTRAN-ról nem beszélhetünk úgy, mint egy megvalósított gépi reprezentációról, egy IBM FORTRAN nyelvről. A különböző fordítóprogramok az IBM FORTRAN különböző változatait ismerik.

A FORTRAN G, H és EXTENDED H fordítók által megvalósított gépi reprezentációk a szabványos ASA (jelenlegi nevén ANSI; American National Standard Institute) FORTRAN nyelv bővítései. A legtöbb bővítést az EXTENDED H fordítóprogram által elfogadott nyelv tartalmazza. A H szintű fordító a nyelv tekintetében nem nyújt lényegesen bővebb lehetőségeket, mint a G szintű, azonban a fordítás folyamán bizonyos optimalizálást is elvégez.

Ebben a függelékben az IBM G szintű fordítóprogramja által ismert FORTRAN változat és a szabvány közötti eltéréseket írjuk le.

A függelék megírásánál nem volt célunk a FORTRAN utasítások részletes ismertetése, hiszen ezt a könyv megteszi. Csak azokat az utasításokat tárgyaljuk, amelyek a szabvány-

tól eltérnek. Az utasítások tárgyalása a könyv által meghatározott sorrendben történik. Mindvégig igyekeztünk a könyv szerkezetét követni. Azokról az utasításokról, amelyek nem szerepelnek az ASA FORTRAN-ban, külön cím alatt szólunk. Ezt a címben *-gal jelöljük. Azokat az utasításokat, amelyek valamilyen értelemben bővebbek, mint a szabvány, részletesen leírjuk. Felesleges ismétlésnek tűnhet, hogy a könyvben már részletesen leírt szabályokat ilyenkor újra megfogalmazzuk, de úgy éreztük, hogy az eltérések tömör ismertetése önmagában nem mindig lenne elegendő. Ezekben az esetekben az eltérést dőlt betűvel való írásmóddal hangsúlyozzuk. A csak a gépi reprezentáció szintjén eltérő utasítások leírásánál apróbetűs írásmódot alkalmaztunk.

2. Bővítések az ASA FORTRAN-hoz képest

A következőkben felsoroljuk a G szintű fordítóprogram által ismert FORTRAN nyelv bővítéseit.

1. COMPLEX*16, REAL*8, INTEGER*2 és LOGICAL*1 típusok.
2. Szövegkonstans idézőjelek között.
3. Hexadecimális konstans.
4. IMPLICIT utasítás.
5. Háromnál több dimenziós tömbök.
6. Általánosított tömbindexek.
7. Változók és tömbök szabványtól eltérő hosszának definiálása explicit típusdeklarációban.
8. Vegyes típusú aritmetikai kifejezések.
9. Többszörös hatványozás a kiszámítás sorrendjét jelző zárójelek nélkül.
10. PAUSE 'UZENET'.
11. READ utasítás ERR és END paramétere.
12. PRINT utasítás.
13. PUNCH utasítás.
14. READ utasítás a kártyaolvasóra.
15. NAMELIST utasítás.
16. T és Z formátum specifikációk.
17. A G formátum specifikáció kibővítése.

18. Adatátvitel közvetlen hozzáférésű adatállományokra: DEFINE FILE, READ, WRITE és FIND utasítások.
19. Explicit típusdeklaráló utasításban kezdőérték adás.
20. Szabványtól eltérő hossz definiálása FUNCTION utasításban.
21. FUNCTION szegmensen belül a függvényazonosító explicit típusdeklaráló és IMPLICIT utasításban szerepelhet.
22. Függvényhivatkozás aktuális paramétere lehet Hollerith konstans.
23. ∗, mint a SUBROUTINE utasítás formális paramétere.
24. RETURN *i* utasítás.
25. Név szerinti paraméterátvétel (formális paraméterek "/" jelek közé zárva is megadhatók).
26. ENTRY utasítás.
27. Utasítás címke, mint a CALL utasítás aktuális paramétere.
28. Aktuális dimenzió átadása COMMON mezőben.

Az EXTENDED H fordítóprogram még további bővítéseket is megenged. Ezekkel a továbbiakban részletesen nem foglalkozunk, itt csak röviden felsoroljuk a legfontosabbakat.

1. Nagy pontosságú aritmetika. Típusdeklarációval megadhatunk REAL∗16 és COMPLEX∗32 típusú változókat. Ezzel közelítőleg 34–35 decimális jegy pontosság érhető el.
2. GENERIC utasítás. Hatására a standard függvények típusa automatikusan igazodik az argumentum típusához.
3. A REAL∗16 és COMPLEX∗32 típusoknak megfelelő standard függvények.
4. Listavezérelt adatátvitel. Egyszerűsíti az adatok beolvasását (a FORMAT utasítás elhagyható).
5. Aszinkron beolvasás/kiírás. A tár és a háttértár között gyorsabb átvitelt biztosít. (Amíg az adatátvitel megtörténik, más programutasítások végrehajthatók.)
6. Új formátum specifikáció a négyszeres pontosságú adatok átvitelére.
7. Az EXTERNAL utasítás kiterjesztése.

ARITMETIKAI ÉS LOGIKAI KIFEJEZÉSEK. RELÁCIÓK

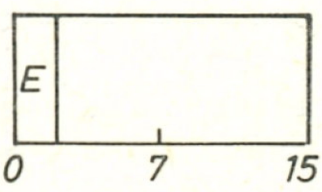
1. Értéktípusok

Az IBM gépek bájt szervezésűek. Az adatok ábrázolása az IBM gépeken a következők szerint történik:

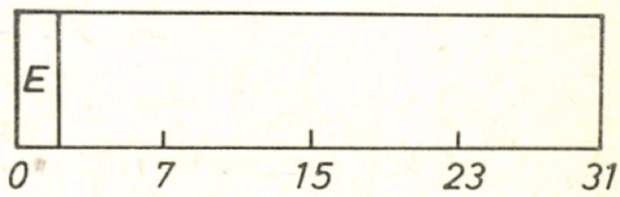
bájt – a központi tár legkisebb címezhető egysége;
félszó – két egymás mellett elhelyezkedő bájt, ahol az első bájt címe páros;
szó – négy egymás mellett elhelyezkedő bájt, ahol az első bájt címe néggyel osztható;
duplaszó – nyolc egymás mellett elhelyezkedő bájt, ahol az első bájt címe nyolccal osztható.

A számábrázolási rendszer a tár bájtokra való felosztásán alapul.

Egész számokat ábrázolhatunk egy félszóban vagy szóban. A legmagasabb helyiértékű bit adja meg az ábrázolt szám előjelét; ha ez a bit Ø, akkor a szám pozitív, ha 1, akkor negatív.

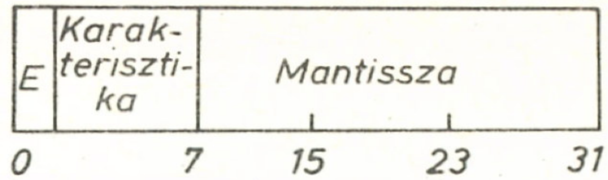


18. ábra.
Egész szám ábrázolása félszóban

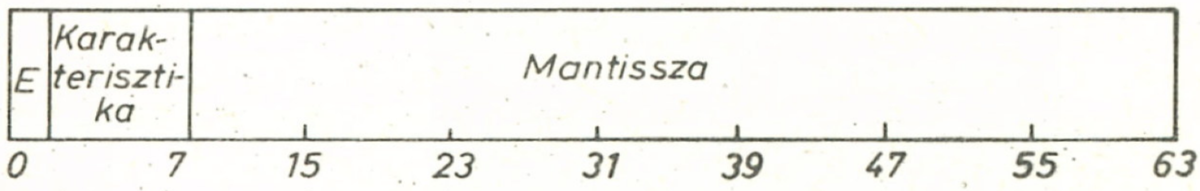


19. ábra.
Egész szám ábrázolása szóban

Valós számokat egy szóban vagy egy duplaszóban ábrázolhatunk. Mindkét esetben a szám karakterisztikája 7 biten helyezkedik el. A szám mantisszája attól függően, hogy szóban vagy duplaszóban ábrázoljuk, 24 bit (3 bájt) vagy 56 bit (7 bájt) lehet. A szám előjelét a legmagasabb helyiértékű bit határozza meg.



20. ábra. Valós szám ábrázolása szóban



21. ábra. Kétszeres pontosságú valós szám ábrázolása duplaszóban

Ennek megfelelően az IBM FORTRAN-*ban lehet 1, 2, 4 és 8 bájt hosszúságú változókat definiálni.*

Minden változótípushoz tartozik egy szabványos hossz-specifikáció, amely megadja a változó bájtokban kifejezett hosszát. Ez a hossz határozza meg a változókban ábrázolható számok nagyságát és pontosságát. A változók szabványos hosszát explicit típusdeklarációval vagy IMPLICIT utasítással felülbírálhatjuk (l. a függelék Negyedik fejezetének 3. alfejezetét).

11. táblázat

A változók típusa és hossza

Típus	Szabványos hossz bájtban	Ábrázolható		Lehetséges hossz bájtban	Ábrázolható	
		nagyság	pontosság		nagyság	pontosság
INTEGER	4	2^{31}—1		2	2^{15}—1	
REAL	4	$\sim 10^{-78}$-tól $\sim 10^{75}$-ig	6—7 decimális jegy	8	$\sim 10^{-78}$-tól $\sim 10^{75}$-ig	15—16 decimális jegy
COMPLEX	8	$\sim 10^{-78}$-tól $\sim 10^{75}$-ig	6—7 decimális jegy	16	$\sim 10^{-78}$-tól $\sim 10^{75}$-ig	15—16 decimális jegy
LOGICAL	4			1		

A változók lehetséges típusát és hosszát, valamint a bennük ábrázolható számok nagyságát és pontosságát a 11. táblázat tartalmazza.

A 8 bájt hosszúságú, valós típusú mennyiségeket kétszeres pontosságúnak (DOUBLE PRECISION) is nevezzük.

A szabványhoz képest az IBM FORTRAN további 3 változótípust tartalmaz: az 1 bájt hosszúságú logikai, a 2 bájt hosszúságú egész és a 16 bájt hosszúságú, kétszeres pontosságú komplex típust.

Egy 16 bájt hosszúságú komplex mennyiség két darab kétszeres pontosságú valós mennyiségből áll.

2. Számok

Az IBM FORTRAN-*ban ötféle típusú számkonstans van.*

Az egész típusú (INTEGER) konstans hossza 4 bájt.

A valós típusú (REAL) konstans hossza 4 bájt.

A kétszeres pontosságú, valós típusú (DOUBLE PRECISION) konstans hossza 8 bájt.

A komplex típusú (COMPLEX) konstans zárójelbe zárt, egymástól vesszővel elválasztott, két valós típusú konstansból áll. Hossza lehet 8 vagy 16 bájt, attól függően, hogy a számpárt egyszeres pontosságú vagy kétszeres pontosságú valós számokkal adtuk meg.

Példa kétszeres pontosságú komplex számra:

(Ø.12DØ,0.31415926535898D1)

A *hexadecimális konstans* Z betűvel kezdődik, amelyet hexadecimális számjegyek követnek. (Hexadecimális számjegyek: a számjegyek Ø-tól 9-ig és az ABC betűi A-tól F-ig.)

Egy bájtban 2 hexadecimális jegyet lehet tárolni. Ha páratlan számú hexadecimális jegyből álló konstanst adtunk meg, a szám balról (legmagasabb helyérték) egy hexadecimális Ø-val egészül ki.

Példa hexadecimális konstansra:

Z1C6A8Ø19

Z66ABCDEF

Hexadecimális konstanst csak kezdőérték adó utasításban (l. a függelék Negyedik fejezetének 5. alfejezetét) lehet megadni. Egy hexadecimális konstans megadható számjegyeinek számát annak a változónak a hossza határozza meg, amelynek kezdeti értéket adtunk.

A változó hossza bájtban	A maximálisan megadható hexadecimális számjegyek száma
8	16
4	8
2	4
1	2

Ha a megadott konstans számjegyeinek száma nagyobb a megengedettnél, a legnagyobb helyiértékű jegyek elvesznek, ha pedig a számjegyek száma kisebb a megengedettnél, a legnagyobb helyiértékű jegy előtt Ø-kal egészül ki. Pl. legyen a változó hossza 2 bájt.

Ekkor a

Z6677A konstansból Z677A,
ZABC konstansból ZØABC lesz.

3. Változók, azonosítók

Az IBM FORTRAN nyelv a szimbolikus nevekben a $ karaktert is megengedi és ezt az ABC-hez tartozó karakternek tekinti. Pl.

```
$PENZ
VAL$1
```

megengedett azonosítók.

4. Aritmetikai műveletek

Az aritmetikai műveletek egész, valós, kétszeres pontosságú és komplex mennyiségek között végezhetők. Az eredmény típusát és hosszát az operandusok típusa és hossza határozza meg.

A négy alapművelet eredményét a 12. táblázatban foglaltuk össze. A táblázatban a típusjelölés alatt zárójelben a hosszt tüntettük fel.

12. táblázat

Az alapműveletek eredményének típusai

Második operandus / Első operandus	INTEGER (2)	INTEGER (4)	REAL (4)	REAL (8)	COMPLEX (8)	COMPLEX (16)
INTEGER (2)	INTEGER (2)	INTEGER (4)	REAL (4)	REAL (8)	COMPLEX (8)	COMPLEX (16)
INTEGER (4)	INTEGER (4)	INTEGER (4)	REAL (4)	REAL (8)	COMPLEX (8)	COMPLEX (16)
REAL (4)	REAL (4)	REAL (4)	REAL (4)	REAL (8)	COMPLEX (8)	COMPLEX (16)
REAL (8)	REAL (8)	REAL (8)	REAL (8)	REAL (8)	COMPLEX (16)	COMPLEX (16)
COMPLEX (8)	COMPLEX (8)	COMPLEX (8)	COMPLEX (8)	COMPLEX (16)	COMPLEX (8)	COMPLEX (16)
COMPLEX (16)	COMPLEX (16)	COMPLEX (16)	COMPLEX (16)	COMPLEX (16)	COMPLEX (16)	COMPLEX (16)

A 12. táblázat alapján kimondhatjuk azt a szabályt, hogy a műveletek eredményének hosszát a résztvevő operandusok hossza közül a nagyobbik határozza meg.

A hatványozásra a következő szabályok érvényesek:

1. REAL (4 bájt) és DOUBLE PRECISION (8 bájt) operandusoknak INTEGER (2 bájt vagy 4 bájt), REAL (4 bájt), DOUBLE PRECISION (8 bájt) típusú kitevőjük lehet.
2. INTEGER (2 bájt vagy 4 bájt) operandusoknak INTEGER (2 bájt vagy 4 bájt), REAL (4 bájt) vagy DOUBLE PRECISION (8 bájt) típusú kitevőjük lehet.
3. Komplex mennyiség csak egész kitevőre emelhető.
4. A hatványozás eredményének típusa és hossza a két operandus típusától és hosszától függ. Az eredmény típusa és hossza a 12. táblázatból olvasható ki.
5. Negatív mennyiség csak egész kitevőre emelhető.
6. Zérust csak pozitív egész kitevőre lehet emelni.

A következő példa bemutatja, hogy a különböző típusú és hosszúságú operandusokkal végzett aritmetikai műveletek milyen típusú és hosszúságú eredményre vezetnek.

Legyenek

C	komplex típusú,	16 bájt hosszúságú;
D	valós típusú,	4 bájt hosszúságú;
I	egész típusú,	4 bájt hosszúságú;
J, K	egész típusú,	2 bájt hosszúságú

változók. Tekintsük az

$$\frac{IJ}{D^K}+C$$

kifejezést. Ez FORTRAN alakban írva:

```
I*J/D**K+C
```

A kifejezés kiszámítása során a következő részeredményeket kapjuk:

I*J	eredménye → R1	4 bájt hosszúságú, egész;
D**K	eredménye → R2	4 bájt hosszúságú, valós;
R1/R2	eredménye → R3	4 bájt hosszúságú, valós;
R3+C	eredménye	16 bájt hosszúságú, komplex.

Látható, hogy a számítás lépései során a részkifejezések értékének típusa és hossza más-más lehet.

Noha a különböző típusú mennyiségekkel végzett aritmetikai műveletek megengedettek, célszerű mégis figyelmet fordítani az operandusok típusára és hosszára. A következő példában egy rossz gyakorlatra szeretnénk felhívni a figyelmet. Legyen C egy 16 bájt hosszúságú, komplex típusú változó. A

2./3.*C

kifejezés kiszámításakor a 2. és 3. konstansokat a program 4 bájt hosszúságú valós mennyiségekként kezeli. Hányadosuk szintén 4 bájt hosszúságú lesz, értéke Ø.6666667. Ez az érték átalakul kétszeres pontosságúvá (a hiányzó értékes jegyek Ø-kkal egészülnek ki) és ez szorzódik össze a C változó valós és képzetes részével, elrontva annak pontosságát. A helyes megoldás:

2.DØ/3.DØ*C

5. Aritmetikai kifejezések

Az aritmetikai kifejezések kiértékelésekor a relatív értelemben vett precedenciaszabály érvényes.

Egyenlő precedenciájú műveletek kiszámítása a *hatványozás kivételével* balról jobbra történik.

Lényeges eltérés az ASA FORTRAN-tól: *egymás után következő hatványozások kiszámítása jobbról balra történik!*

Például az

A**B**C

kifejezés kiszámításának sorrendje:

B**C eredménye → R

A**R a végeredmény $A^{(B^C)}$

Az IBM FORTRAN a szabványhoz képest *több standard függvényt* definiál. Eltérés van abban is, hogy mely függvények külső és melyek belső függvények. A standard belső függvényeket a 13. táblázat, a külső függvényeket a 14. táblázat tartalmazza.

13. táblázat

Belső standard függvények

Elnevezés	Függvény-azonosító	Definíció	Argumentum száma	Argumentum típusa	Argumentum értelmezési tartomány	A függvény típusa
Abszolút érték	IABS	$y = \|x\|$	1	INTEGER*4	tetszőleges egész argumentum	INTEGER*4
	ABS		1	REAL*4	tetszőleges valós argumentum	REAL*4
	DABS		1	REAL*8		REAL*8
Csonkítás vagy abszolút lefelé kerekítés	AINT	$y = \text{sign}(x) \cdot n$; ahol n a legnagyobb egész $\leqq \|x\|$	1	REAL*4	tetszőleges valós argumentum	REAL*4
	INT		1	REAL*4		INTEGER*4
	IDINT		1	REAL*8		INTEGER*4
Maradékképzés	MOD	$y = x_1 \cdot (\text{modulo } x_2)$	2	INTEGER*4	$x_2 \neq \emptyset$	INTEGER*4
	AMOD		2	REAL*4		REAL*4
	DMOD [1]		2	REAL*8		REAL*8
Konverzió egészből valósba	FLOAT		1	INTEGER*4	tetszőleges egész argumentum	REAL*4
	DFLOAT [2]		1	INTEGER*4		REAL*8
Konverzió valósból egészbe	IFIX	$y = \text{sign}(x) \cdot n$; ahol n a legnagyobb egész $\leqq \|x\|$	1	REAL*4	tetszőleges valós argumentum	INTEGER*4
	HFIX [2]		1	REAL*4		INTEGER*2
Előjelezés	ISIGN	$y = \text{sign}(x_2) \cdot \|x_1\|$	2	INTEGER*4	$x_2 \neq \emptyset$	INTEGER*4
	SIGN		2	REAL*4		REAL*4
	DSIGN		2	REAL*8		REAL*8

13. táblázat folytatása

Elnevezés	Függvény-azonosító	Definíció	Argumentum száma	Argumentum típusa	Argumentum értelmezési tartomány	A függvény típusa
Pozitív differencia	IDIM	$y = x_1 - \min(x_1, x_2)$	2	INTEGER*4	tetszőleges egész argumentumok	INTEGER*4
	DIM		2	REAL*4	tetszőleges valós argumentumok	REAL*4
Konverzió kétszeres pontosságúból valósba	SNGL		1	REAL*8	tetszőleges valós argumentum	REAL*4
Valós rész képzés komplex számból	REAL		1	COMPLEX*8	tetszőleges komplex argumentum	REAL*4
Képzetes rész képzés komplex számból	AIMAG		1	COMPLEX*8	tetszőleges komplex argumentum	REAL*4
Konverzió valósból kétszeres pontosságúba	DBLE		1	REAL*4	tetszőleges valós argumentum	REAL*8
Komplex szám képzés	CMPLX	$y = x_1 + x_2 i$	2	REAL*4	tetszőleges valós argumentumok	COMPLEX*8
	DCMPLX [2]		2	REAL*8		COMPLEX*16
Komplex konjugált képzés	CONJG	$y = x_1 - x_2 i$ ($\arg = x_1 + x_2 i$)	1	COMPLEX*8	tetszőleges komplex argumentum	COMPLEX*8
	DCONJG [2]		1	COMPLEX*16		COMPLEX*16

1. az ASA FORTRAN-ban külső függvény
2. nincs az ASA FORTRAN-ban

Külső standard

1. Exponenciális és

Elnevezés	Függvényazonosító	Definíció	
Természetes logaritmus	ALOG	$y = \ln(x)$	
	DLOG		
	CLOG	$y = \ln(z)$ főértéke [2] ($z = x_1 + x_2 i$)	
	CDLOG [1]		
Tizes alapú logaritmus	ALOG1Ø	$y = \log_{1Ø}(x)$	
	DLOG1Ø		
Exponenciális függvény	EXP	$y = e^x$	
	DEXP		
	CEXP	$y = e^z$ ($z = x_1 + x_2 i$)	
	CDEXP [1]		
Négyzetgyökvonás	SQRT	$y = \sqrt{x}$ vagy $y = x^{1/2}$	
	DSQRT		
	CSQRT	$y = \sqrt{z}$ ($z = x_1 + x_2 i$)	
	CDSQRT [1]		

1. nincs az ASA FORTRAN-ban
2. az eredményben a képzetes rész $-\pi$ és π közé esik, pontosan: $-\pi < y_2 \leqq \pi$ teljesül,
3. egyszeres pontosság esetén $r = 16^{63}\ (1-16^{-6})$, kétszeres pontosság esetén $r = 16^{63}$

14. táblázat

függvények

logaritmikus függvények

Argumentum			Függvény	
száma	típusa	értelmezési tartomány	típusa	értékkészlete
1	REAL*4	$x > \emptyset$	REAL*4	$-18\emptyset.218 \leqq y \leqq 174.673$
1	REAL*8		REAL*8	
1	COMPLEX*8	$z \neq \emptyset.\emptyset + \emptyset.\emptyset i$	COMPLEX*8	$-18\emptyset.218 \leqq y_1 \leqq 175.\emptyset 21$ $-\pi < y_2 \leqq \pi$
1	COMPLEX*16		COMPLEX*16	
1	REAL*4	$x > \emptyset$	REAL*4	$-78.268 \leqq y \leqq 75.859$
1	REAL*8		REAL*8	
1	REAL*4	$-18\emptyset.218 \leqq x \leqq$ $\leqq 174.673$	REAL*4	$\emptyset \leqq y \leqq r$ [3]
1	REAL*8		REAL*8	
1	COMPLEX*8	$x_1 \leqq 174.673$ $\mid x_2 \mid < (2^{18} \cdot \pi)$	COMPLEX*8	$-r = y_1, y_2 \leqq r$ [3]
1	COMPLEX*16	$x_1 \leqq 174.673$ $\mid x_2 \mid < (2^{50} \cdot \pi)$	COMPLEX*16	
1	REAL*4	$x \geqq \emptyset$	REAL*4	$\emptyset \leqq y \leqq r^{1/2}$ [3]
1	REAL*8		REAL*8	
1	COMPLEX*8	tetszőleges komplex argumentum	COMPLEX*8	$\emptyset \leqq y \leqq 1.\emptyset 987 \cdot$ $\cdot (r^{1/2})$ [3] $\mid y_2 \mid \leqq 1.\emptyset 987 \cdot (r^{1/2})$
1	COMPLEX*16		COMPLEX*16	

azonban ha $x_1 < \emptyset$ és $x_2 = \emptyset$, akkor $y_2 = -\pi$

$(1-16^{-14})$

2. Trigonometrikus és

Elnevezés	Függvényazonosító	Definíció	
Árkusz szinusz	ARSIN [1]	$y = \arcsin(x)$	
	DARSIN [1]		
Árkusz koszinusz	ARCOS [1]	$y = \arccos(x)$	
	DARCOS [1]		
Árkusz tangens	ATAN	$y = \mathrm{arctg}(x)$	
	DATAN		
	ATAN2	$y = \mathrm{arctg}\left(\frac{x_1}{x_2}\right)$	
	DATAN2		
Szinuszfüggvény	SIN	$y = \sin(x)$	
	DSIN		
	CSIN	$y = \sin(z)$ $(z = x_1+x_2i)$	
	CDSIN [1]		
Koszinuszfüggvény	COS	$y = \cos(x)$	
	DCOS		
	CCOS	$y = \cos(z)$ $(z = x_1+x_2i)$	
	CDCOS [1]		
Tangensfüggvény	TAN [1]	$y = \mathrm{tg}(x)$	
	DTAN [1]		
Kotangensfüggvény	COTAN [1]	$y = \mathrm{ctg}(x)$	
	DCOTAN [1]		
Szinusz hiperbolikusz	SINH [1]	$y = \frac{e^x - e^{-x}}{2}$	
	DSINH [1]		
Koszinusz hiperbolikusz	COSH [1]	$y = \frac{e^x + e^{-x}}{2}$	
	DCOSH [1]		
Tangens hiperbolikusz	TANH	$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	
	DTANH [1]		

1. nincs az ASA FORTRAN-ban
2. egyszeres pontosság esetén $r = 16^{63} \cdot (1-16^{-6})$, kétszeres pontosság esetén
3. n páratlan szám
4. n egész szám

14. táblázat 1. folytatása

hiperbolikus függvények

Argumentum			Függvény	
száma	típusa	értelmezési tartomány	típusa	értékkészlete
1	REAL*4	$\lvert x \rvert \leqq 1$	REAL*4	$-\frac{\pi}{2} \leqq y \leqq \frac{\pi}{2}$
1	REAL*8		REAL*8	
1	REAL*4	$\lvert x \rvert \leqq 1$	REAL*4	$Ø \leqq y \leqq \pi$
1	REAL*8		REAL*8	
1	REAL*4	tetszőleges valós argumentum	REAL*4	$-\frac{\pi}{2} \leqq y \leqq \frac{\pi}{2}$
1	REAL*8		REAL*8	
2	REAL*4	tetszőleges valós argumentum, kivéve $x_1 = x_2 = Ø$	REAL*4	$-\pi = y = \pi$
2	REAL*8		REAL*8	
1	REAL*4 (radiánban)	$\lvert x \rvert < (2^{18} \cdot \pi)$	REAL*4	$-1 \leqq y \leqq 1$
1	REAL*8 (radiánban)	$\lvert x \rvert < (2^{50} \cdot \pi)$	REAL*8	
1	COMPLEX*8	$\lvert x_1 \rvert < (2^{18} \cdot \pi)$ $\lvert x_2 \rvert \leqq 174.673$	COMPLEX*8	$-r \leqq y_1, y_2 \leqq r$ [2]
1	COMPLEX*16	$\lvert x_1 \rvert < (2^{5Ø} \cdot \pi)$ $\lvert x_2 \rvert \leqq 174.673$	COMPLEX*16	
1	REAL*4 (radiánban)	$\lvert x \rvert < (2^{18} \cdot \pi)$	REAL*4	$-1 \leqq y \leqq 1$
1	REAL*8 (radiánban)	$\lvert x \rvert < (2^{5Ø} \cdot \pi)$	REAL*8	
1	COMPLEX*8	$\lvert x_1 \rvert < (2^{18} \cdot \pi)$ $\lvert x_2 \rvert \leqq 174.673$	COMPLEX*8	$-r \leqq y_1, y_2 \leqq r$ [2]
1	COMPLEX*16	$\lvert x_1 \rvert < (2^{5Ø} \cdot \pi)$ $\lvert x_2 \rvert \leqq 174.673$	COMPLEX*16	
1	REAL*4 (radiánban)	$\lvert x \rvert < (2^{18} \cdot \pi)$ [3] $\lvert x \rvert \neq \pi/2 \cdot n$	REAL*4	$-r \leqq y \leqq r$ [2]
1	REAL*8 (radiánban)	$\lvert x \rvert < (2^{5Ø} \cdot \pi)$ [3] $\lvert x \rvert \neq \pi/2 \cdot n$	REAL*8	
1	REAL*4 (radiánban)	$\lvert x \rvert < (2^{18} \cdot \pi)$ [4] $\lvert x \rvert \neq \pi \cdot n$	REAL*4	$-r \leqq y \leqq r$ [2]
1	REAL*8 (radiánban)	$\lvert x \rvert < (2^{5Ø} \cdot \pi)$ [4] $\lvert x \rvert \neq \pi \cdot n$	REAL*8	
1	REAL*4	$\lvert x \rvert < 175.366$	REAL*4	$-r \leqq y \leqq r$ [2]
1	REAL*8		REAL*8	
1	REAL*4	$\lvert x \rvert < 175.366$	REAL*4	$1 \leqq y \leqq r$ [2]
1	REAL*8		REAL*8	
1	REAL*4	tetszőleges valós argumentum	REAL*4	$-1 \leqq y \leqq 1$
1	REAL*8		REAL*8	

$r = 16^{63} \cdot (1 - 16^{-14})$

3. Egyéb matematikai

Elnevezés	Függvényazonosító		Definíció	
Abszolút érték	CABS		$y = \lvert z \rvert = (x_1^2 + x_2^2)^{1/2}$	
	CDABS	1		
Hibafüggvény	ERF	1	$y = \frac{2}{\sqrt{\pi}} \int_0^x e^{-u^2} du$	
	DERF	1		
	ERFC	1	$y = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-u^2} du$	
	DERFC	1		
Gammafüggvény	GAMMA	1	$y = \int_\emptyset^\infty u^{x-1} e^{-u} du$	
	DGAMMA	1		
	ALGAMA	1	$y = \ln \int_\emptyset^\infty u^{x-1} e^{-u} du$	
	DLGAMA	1		
A legnagyobb argumentum kiválasztása	MAXØ	3	$y = \max(x_1, x_2, \ldots, x_n)$	
	AMAXØ	3		
	MAX1	3		
	AMAX1	3		
	DMAX1	3		
A legkisebb argumentum kiválasztása	MINØ	3	$y = \min(x_1, x_2, \ldots, x_n)$	
	AMINØ	3		
	MIN1	3		
	AMIN1	3		
	DMIN1	3		

1. nincs az ASA FORTRAN-ban
2. egyszeres pontosság esetén $r = 16^{63} \cdot (1 - 16^{-6})$, kétszeres pontosság esetén
3. az ASA FORTRAN-ban belső függvény

14. táblázat 2. folytatása

függvények

Argumentum			Függvény	
száma	típusa	értelmezési tartomány	típusa	értékkészlete
1	COMPLEX*8	tetszőleges komplex argumentum	REAL*4	$\emptyset \leqq y_1 \leqq r$ [2] $y_2 = \emptyset$
1	COMPLEX*16		REAL*8	
1	REAL*4	tetszőleges valós argumentum	REAL*4	$-1 \leqq y \leqq 1$
1	REAL*8		REAL*8	
1	REAL*4		REAL*4	$\emptyset \leqq y \leqq 2$
1	REAL*8		REAL*8	
1	REAL*4	$2^{-252} < x < 57.5744$	REAL*4	$\emptyset.88560 \leqq y \leqq r$ [2]
1	REAL*8		REAL*8	
1	REAL*4	$\emptyset < x < 4.2937 \cdot 1\emptyset^{73}$	REAL*4	$-\emptyset.12149 \leqq y \leqq r$ [2]
1	REAL*8		REAL*8	
≧2	INTEGER*4	tetszőleges egész argumentumok	INTEGER*4	
≧2	INTEGER*4		REAL*4	
≧2	REAL*4	tetszőleges valós argumentumok	INTEGER*4	
≧2	REAL*4		REAL*4	
≧2	REAL*8		REAL*8	
≧2	INTEGER*4	tetszőleges egész argumentumok	INTEGER*4	
≧2	INTEGER*4		REAL*4	
≧2	REAL*4	tetszőleges valós argumentumok	INTEGER*4	
≧2	REAL*4		REAL*4	
≧2	REAL*8		REAL*8	

$r = 16^{63} \cdot (1 - 16^{-14})$

6. Relációk

A relációk kiértékelése után mindig egy 4 bájt hosszúságú logikai típusú érték keletkezik.

Relációba hozhatunk egymással

— két egész kifejezést;

— két valós kifejezést;

— *egy valós és egy egész kifejezést.*

A relációba hozott két aritmetikai kifejezés értéke különböző hosszúságú lehet. Pl., ha I és K egész változók hossza 2 bájt, a

K+I.LT. 2.ØDØ

reláció kétszeres pontosságú valós mennyiséget egy 2 bájt hosszúságú mennyiséggel hasonlít össze.

7. Logikai változók, műveletek, kifejezések

A logikai változók szabványos hossza 4 bájt, de *lehetnek 1 bájt hosszúságúak is.* A 15. táblázat a logikai műveletek eredményének hosszát mutatja.

15. táblázat

A logikai műveletek eredményének hossza

Második operandus / Első operandus	LOGICAL (1)	LOGICAL (4)
LOGICAL (1)	(1)	(4)
LOGICAL (4)	(4)	(4)

Harmadik fejezet

VÉGREHAJTHATÓ UTASÍTÁSOK

1. Az utasítások típusai az IBM FORTRAN nyelvben

Az IBM FORTRAN-ban az utasítások típusai nem térnek el a szabványtól.

2. Értékadó utasítások

a) Logikai értékadó utasítás

A logikai értékadó utasításban különböző hosszúságú logikai változók szerepelhetnek, a konverzió mindig elvégezhető. Pl., ha az L logikai változó hossza 1 bájt, K és C hossza pedig 4 bájt, a következő értékadó utasítások helyesek:

```
L = .NOT.K
K = L.AND.C
```

b) Aritmetikai értékadó utasítás

A V = *e* aritmetikai értékadó utasítás konverziós szabályait a 16. táblázat tartalmazza. A táblázatban az egyszerű értékadást E-vel jelöltük.

A táblázatból kiolvasható, hogy ha az értékadó utasításban az egyenlőségjel egyik oldalán komplex típusú mennyiség szerepel, de a másikon nem, a konverzió a szabványtól eltér. Pl. legyen C komplex típusú változó. A

```
C = A*B
```

értékadó utasításban C képzetes része Ø lesz, az

```
A = C*B
```

16. táblázat

Az aritmetikai értékadás konverziós szabályai

V \ e	INTEGER*2 INTEGER*4	REAL*4	REAL*8	COMPLEX*8	COMPLEX*16
INTEGER*2 INTEGER*4	E	IFIX	SNGL és IFIX	valós rész [1] IFIX	valós rész [1] SNGL és IFIX
REAL*4	FLOAT	E	SNGL	valós rész [1] E	valós rész [1] SNGL
REAL*8	FLOAT és DBLE	DBLE	E	valós rész [1] DBLE	valós rész [1] E
COMPLEX*8	FLOAT [2]	E [2]	SNGL [2]	E	SNGL valós és SNGL képzetes
COMPLEX*16	FLOAT és [2] DBLE	DBLE [2]	E [2]	DBLE valós és DBLE képzetes	E

1. a képzetes rész nem vesz részt az értékadásban
2. a valós rész veszi fel *e* értékét; a képzetes rész Ø lesz

utasításban pedig a C*B komplex mennyiség képzetes része nem vesz részt az értékadásban. A változó a valós rész értékét veszi fel.

Felhívjuk a figyelmet arra, hogy a

C = (A, B)

alakú kifejezés, ahol C komplex típusú, nem használható! Egy komplex mennyiséget két valós mennyiségből a CMPLX standard függvény segítségével képezhetünk. Tehát az előző értékadás helyesen:

C = CMPLX (A, B)

3. Vezérlő utasítások és címkék. ASSIGN utasítás

Az ASSIGN utasítás alakja

ASSIGN *l* TO *i*,

ahol *l* egy végrehajtható utasítás címkéje; *i* egy INTEGER (4 bájt hosszúságú) típusú változó azonosítója.
Megjegyzés: *i* csak *szabványos* hosszúságú, egész típusú változó lehet.

a) Feltétlen vezérlésátadás. A feltétlen és a kijelölt GO TO utasítás

A feltétlen és a kijelölt GO TO utasítások után címkével ellátott utasításnak kell következnie. (Ugyanis a címkézetlen utasítás végrehajtására soha nem kerülne sor.) Ha ezt a szabályt nem tartjuk be, a fordítóprogram informatív hibaüzenetet ad.

b) Elágaztatás előjel és nulla szerint. Az aritmetikai IF utasítás

Az aritmetikai IF utasítás után címkével ellátott utasításnak kell következnie. Ha ezt a szabályt nem tartjuk be, a fordítóprogram informatív hibaüzenetet ad.

c) Feltételes utasítás. A logikai IF utasítás

A logikai IF utasítás az IBM FORTRAN-ban megegyezik a szabvánnyal.

d) Többirányú elágaztatás. A kiszámított GO TO utasítás

A kiszámított GO TO utasítás alakja:

$$\text{GO TO}\,(l_1, l_2, \ldots, l_n), i,$$

ahol l_i egy végrehajtható utasítás címkéje; i egy INTEGER (2 vagy 4 bájt hosszúságú) típusú változó azonosítója. i-nek értéket kell adni, mielőtt a GO TO utasításra sor kerülne.

Ha i értéke kívül esik az $1 \leqq i \leqq n$ tartományon, a GO TO utasítás után következő utasítás fog végrehajtódni.

4. Indexes változók, tömbök. Indexkifejezések

Egy tömb dimenzióinak száma legfeljebb 7 lehet. A tömb elemei oszlopfolytonosan helyezkednek el a központi tárban.

A FORTRAN szabvány által megadott lehetőségeken túlmenően a tömbelemek kijelölésére *általánosított indexkifejezéseket* használhatunk.

Az indexkifejezésekre a következő szabályoknak kell teljesülniük:

1. Az indexkifejezés tartalmazhat aritmetikai kifejezést, amelyben a +, −, *, /, ** aritmetikai operátorok szerepelnek, továbbá függvényhivatkozásokat és tömbelemeket.

2. Az aritmetikai kifejezésben egész és valós típusú változók szerepelhetnek. A kifejezés kiszámítása az aritmetikai műveletekre vonatkozó szabályoknak (l. a 12. táblázatot) megfelelően történik. Ha a kiszámított kifejezés értéke nem egész szám, az indexkifejezés értéke a valós mennyiség egész része lesz.

3. A kiszámított indexkifejezés értéke nem lehet zérus vagy negatív.

Példák általánosított indexkifejezésekre:

TOMB (I, J/4*K**2)
MATRIX (J+TOMB (K,L), IABS(M))
ARRAY (I—6)

5. Ciklusutasítások és használatuk

Az egymásba skatulyázott DO utasítások maximális száma 25 lehet.

6. Megállító utasítások. A PAUSE és a STOP utasítás

A PAUSE utasítás alakjai:

PAUSE
PAUSE *n*
PAUSE 'UZENET'

ahol *n* legfeljebb 5 decimális számjegyből állhat; 'UZENET' egy idézőjelek közé zárt szövegkonstans. Ha a szövegkonstans karakterei között idézőjel is szerepel, két egymást követő idézőjelet kell írni, pl.

PAUSE 'DON''T WORRY'

Az utasítás hatására a program futása felfüggesztődik és az operátori konzolon a

PAUSE Ø
PAUSE *n*
PAUSE UZENET

üzenetek egyike jelenik meg az utasítás alakjának megfelelően. A számítógép kezelője a programozó utasítása szerint a programot továbbindíthatja.

A STOP utasítás alakjai:

STOP
STOP *n*

ahol *n* legfeljebb 5 decimális számjegyből állhat.

A STOP utasítás hatására a program végrehajtása befejeződik. Ha a STOP *n* formát használtuk, az operátori konzolon a

STOP *n*

üzenet jelenik meg.

Negyedik fejezet

DEKLARATÍV UTASÍTÁSOK (NEMVÉGREHAJTHATÓ UTASÍTÁSOK)

1. A FORTRAN program szerkezete. A szegmensek

Az IBM FORTRAN programok szerkezete nem tér el a szabványtól.

2. A nemvégrehajtható utasítások fajtái. Az utasítások elhelyezése a szegmenseken belül

A nemvégrehajtható utasítások a következők:

1. Szegmensnyitó utasítások: SUBROUTINE, FUNCTION, BLOCK DATA. A főprogramnak nincs szegmensnyitó utasítása. (A MASTER szegmensnyitó utasítás nem használható).

2. Specifikációs utasítások:

a) Típusdeklaráló utasítások: INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL és IMPLICIT.
b) Tömbdeklaráló utasítás: DIMENSION.
c) Közös adatmezőt deklaráló utasítás: COMMON.
d) EQUIVALENCE utasítás.
e) EXTERNAL utasítás.

3. Kezdőérték adó utasítás: DATA.

4. FORMAT utasítás.

5. NAMELIST utasítás.

6. DEFINE FILE utasítás.

7. ENTRY utasítás.

8. Utasításfüggvény.

9. Szegmenszáró utasítás: END.

Egy adott szegmensen belül a nemvégrehajtható utasítások sorrendjére a következő szabályok érvényesek:

a) Első a szegmensnyitó utasítás, ha van.

b) Az IMPLICIT utasításnak (l. a következőkben), ha használjuk, a főprogramban az első utasításnak kell lennie, szubrutinban vagy függvényben rögtön a szegmensnyitó utasítás után kell következnie.

c) Ezután következnek a további deklaráló és kezdőérték adó utasítások. Ezek sorrendjére nézve nincs megszorítás.

3. Típus- és tömbdeklarációk

a) Típusdeklarációk

A programban használt változók típusát megadhatjuk:

— automatikus típusdeklaráció;

— explicit típusdeklaráció;

— és az IMPLICIT utasítás

segítségével.

Egy explicit deklarációs utasítás felülírja az IMPLICIT utasítás hatását, az IMPLICIT utasítás pedig felülbírálja az automatikus típusdeklarációt. Az automatikus típusdeklarációval megadott változók hossza mindig szabványos.

Egy változónak a szabványtól eltérő hosszát csak explicit deklarációval vagy az IMPLICIT utasítással adhatjuk meg.

a1)* IMPLICIT utasítás

Az IMPLICIT utasítással minden olyan változónak, tömbnek és a programozó által definiált függvénynek, amelyek neve egy megadott betűvel kezdődik, típusát és hosszát előre meghatározhatjuk. Ezzel az automatikus típusdeklarációval előírt típusokat és hosszakat felülbírálhatjuk.

Az IMPLICIT utasítás alakja:

IMPLICIT $típus_1 * s_1(a_{11}, a_{12}, \ldots), \ldots, típus_n * s_n(a_{n1}, a_{n2}, \ldots)$

ahol $típus_1$ az INTEGER, REAL, COMPLEX vagy LOGICAL FORTRAN alapszavak egyike.

s_i a változó hosszát adja meg. Nem kötelező kiírni, ha elhagyjuk, a változó szabványos hosszúságú lesz. Az 1, 2, 4, 8 vagy 16

egész számok egyike lehet. a_{ij} az ABC egy betűje vagy az ABC egy tartománya. A tartományt úgy jelöljük, hogy a tartomány első és utolsó betűje közé a „—" (mínusz) jelet tesszük. A tartomány a kezdő és záró betűvel együtt értendő.

Az IMPLICIT utasítás egy programszegmensen belül csak egyszer adható meg, és ha megadjuk, a főprogram első utasítása kell, hogy legyen, a SUBROUTINE vagy FUNCTION programszegmensekben pedig közvetlenül a szegmensnyitó utasítás után kell állnia. Az IMPLICIT utasításban típusként DOUBLE PRECISION-t nem adhatunk meg. Helyette a REAL*8 kétszeres pontosságú valós típust kell használnunk. A DOUBLE PRECISION típus a REAL*8 típussal azonos.

Példák az IMPLICITutasításra:

```
IMPLICIT INTEGER (A—D), COMPLEX*16(K),
+ REAL*8 (O—Z)
```

Az utasítás hatására az A, B, C, D betűvel kezdődő változók vagy tömbök 4 bájt hosszúságú, egész típusúak, a K betűvel kezdődőek 16 bájt hosszúságú, komplex típusúak, az O, P, ..., Z betűvel kezdődőek pedig kétszeres pontosságú, valós típusúak lesznek.

```
IMPLICIT REAL*8(A—H, O—Z)
```

Az előző utasítással a változók kezdőbetűire vonatkozó konvenciót nem bíráljuk felül, azonban minden (explicit módon nem deklarált) valós típusú változóhoz a 8 bájt hosszúságot rendeljük, azaz minden valós változó kétszeres pontosságú lesz.

```
IMPLICIT INTEGER*2 (A—H), REAL*8 (I—K),
+ LOGICAL (L,M,N)
```

Az utasítás hatására minden olyan változó, amely az A, B, ..., H betűk valamelyikével kezdődik, 2 bájt hosszúságú, egész típusú lesz. Minden I, J vagy K betűvel kezdődő kétszeres pontosságú (8 bájt hosszúságú), valós típusú, az L, M, N betűkkel kezdődő változók pedig logikai típusúak lesznek. Mivel az O, P, ..., Z, $ betűkkel kezdődő változókat az IMPLICIT utasítás nem definiálta, ezekre az automatikus típusdeklarálási szabály érvényes (egyszeres pontosságú, 4 bájt hosszúságú, valós típus).

a2) Explicit típusdeklaráció

Az explicit típusdeklaráló utasítás egy-egy konkrét változóra, tömbre vagy függvényre vonatkozóan felülbírálja az automatikus típusdeklarációval vagy az IMPLICIT utasítással megadott alaptípusokat és hosszakat. Az automatikus és az IMPLICIT típusdeklaráció az azonosítók kezdőbetűje alapján határozza meg a változó típusát. Az explicit deklarációban névszerint fel kell sorolni azokat a változókat, amelyek típusát elő akarjuk írni.

Alakja:

$$típus*s\ a_1*s_1(k_1)\ /x_1/,\ a_2*s_2(k_2)\ /x_2/,\ \ldots,\ a_n*s_n(k_n)\ /x_n/$$

ahol *típus* az INTEGER, REAL, LOGICAL vagy COMPLEX FORTRAN alapszavak valamelyike lehet.

s és/vagy s_i egy, a *változó típusához tartozó megengedett hosszat specifikál.* Nem kötelező kiírni, ha elhagyjuk, a változó szabványos hosszúságú lesz. Ha a hosszmegadás hiányzik, akkor a *-ot sem szabad kiírnunk.

a_i egy változó, tömb vagy függvény neve.

k_i elhagyható, tömbök deklarációjában a tömb dimenzióját határozza meg. A k_i-k vesszővel elválasztott előjel nélküli egész számok listái, amelyek az indexek maximális értékét adják meg. A lista elemeinek száma legfeljebb 7 lehet. Ha az explicit típusdeklaráló utasítás egy szubrutinban vagy függvényben van megadva, k_i egész típusú, 4 bájt hosszúságú változó lehet (dinamikus indexhatár).

$/x_i/$ nem kötelező, a változónak adott kezdeti érték.

Az explicit típusdeklaráló utasítás használható kezdőérték betöltésére is. Kezdőértékként konstanst vagy vesszőkkel tagolt konstanslistát adhatunk meg.

A két törtvonal közé zárt konstans csak a közvetlenül előtte álló változót, a két törtvonal közé zárt konstanslista az előtte álló tömböt tölti fel. Ugyanannak a konstansnak többszöri előfordulását az $i*$ ismétlési tényezővel jelezhetjük, ahol i pozitív egész szám. A konstansnak ugyanolyan típusúnak kell lennie, mint a változónak, amelynek a kezdőértéket adtuk.

Kivételt képeznek a szövegkonstansok (literálisok) és a hexadecimális konstansok, amelyek bármilyen típusú változó kezdőértékeként megadhatók.

Nem adható kezdeti érték olyan változónak vagy tömbnek, amely blank COMMON-ban szerepel, valamint formális paraméternek és függvénynévnek. Nevezett COMMON-ban szereplő változónak csak BLOCK DATA szegmensben adhatunk explicit típusdeklarációval értéket.

Az explicit típusdeklaráló utasítás használható tömb dimenziójának meghatározására is.

Egy változó, tömb vagy függvény neve csak egy típusdeklaráló utasításban szerepelhet.

Az explicit típusdeklaráló utasításban specifikálhatjuk egy előzetesen COMMON vagy DIMENSION utasításban megadott változó vagy tömb típusát és hosszát.

Példák explicit típusdeklaráló utasításokra:

```
INTEGER*2 A, A1, F2, M*4/256/, Q7
```

hatására az A, az A1, az F2 és a Q7 2 bájt hosszúságú, az M pedig 4 bájt hosszúságú, egész típusú változó lesz. M-nek a 256 kezdeti értéket adtuk.

```
COMPLEX C, D/(2.1, 4.7)/, E*16
```

C, D és E komplex típusú változók. Mivel hosszt nem adtunk meg, C és D hosszúsága 8 bájt (4 bájt a valós rész, 4 bájt a képzetes rész). A D változó valós részének a 2.1, képzetes részének a 4.7 kezdeti értéket adtunk. E hosszúsága 16 bájt lesz. (8 bájt a valós rész, 8 bájt a képzetes rész).

```
REAL A(5,5)/2Ø*6.9E2,5*1.Ø/, B*8(1ØØ)/1ØØ*Ø.ØDØ/
```

Ez az utasítás az A és B valós típusú tömböket deklarálja. Az A tömb 1ØØ bájt helyet foglal el a központi tárban (minden elem 4 bájt hosszúságú), a B tömb pedig 8ØØ bájtot (mivel minden eleme 8 bájt hosszúságú). Az A tömb első 2Ø elemének kezdeti értéke 6.9E2, a fennmaradó 5 elem értéke 1.Ø lesz, végül a B tömb minden eleme a Ø.ØDØ értéket veszi fel.

Az explicit típusdeklaráló utasítások közé tartozik a

DOUBLE PRECISION $a_1(k_1)$, $a_2(k_2)$, ..., $a_n(k_n)$

utasítás is, ahol a_i egy változó, tömb vagy függvényazonosító. k_i elhagyható, tömbök deklarációja esetén a tömb dimenzióját határozza meg. A k_i-k vesszővel elválasztott előjel nélküli egész számok listái, amelyek az indexek maximális értékét adják meg. A listák legfeljebb 7 elemet tartalmazhatnak. Ha az utasítás egy szubrutinban vagy függvényben van megadva, k_i lehet egész típusú, 4 bájt hosszúságú változó is.

A DOUBLE PRECISION utasítás felülbírálja az automatikus típusdeklarációt és az IMPLICIT utasítást. Eredményeként minden deklarált változó kétszeres pontosságú lesz, azaz 8 bájt hosszúságú. A REAL∗8 és a DOUBLE PRECISION típusdeklaráló utasítások hatása azonos. A DOUBLE PRECISION utasítással nem lehet kezdőértéket beállítani.

b) Tömbdeklarációk. A DIMENSION utasítás

A tömbök számára lefoglalandó tárterület nagyságát a DIMENSION utasítással is megadhatjuk. *Az* IBM FORTRAN*-ban a tömbök dimenziójának száma legfeljebb 7 lehet.*

Alakja:

DIMENSION $a_1(k_1)$, $a_2(k_2)$, ..., $a_n(k_n)$

ahol a_i egy tömb azonosítója, k_i vesszővel elválasztott előjel nélküli egész számok listája, amelyek elemei az indexek maximális értékét adják meg. A lista legfeljebb 7 elemet tartalmazhat. Ha a DIMENSION utasítás egy szubrutinban vagy függvényben van megadva, k_i lehet egész típusú, 4 bájt hosszúságú változó is.

Például

```
| |DIMENSION A(1Ø), B(5, 3, 2), C(7, 9, 5, 4, 2)
```

4. Az EQUIVALENCE utasítás

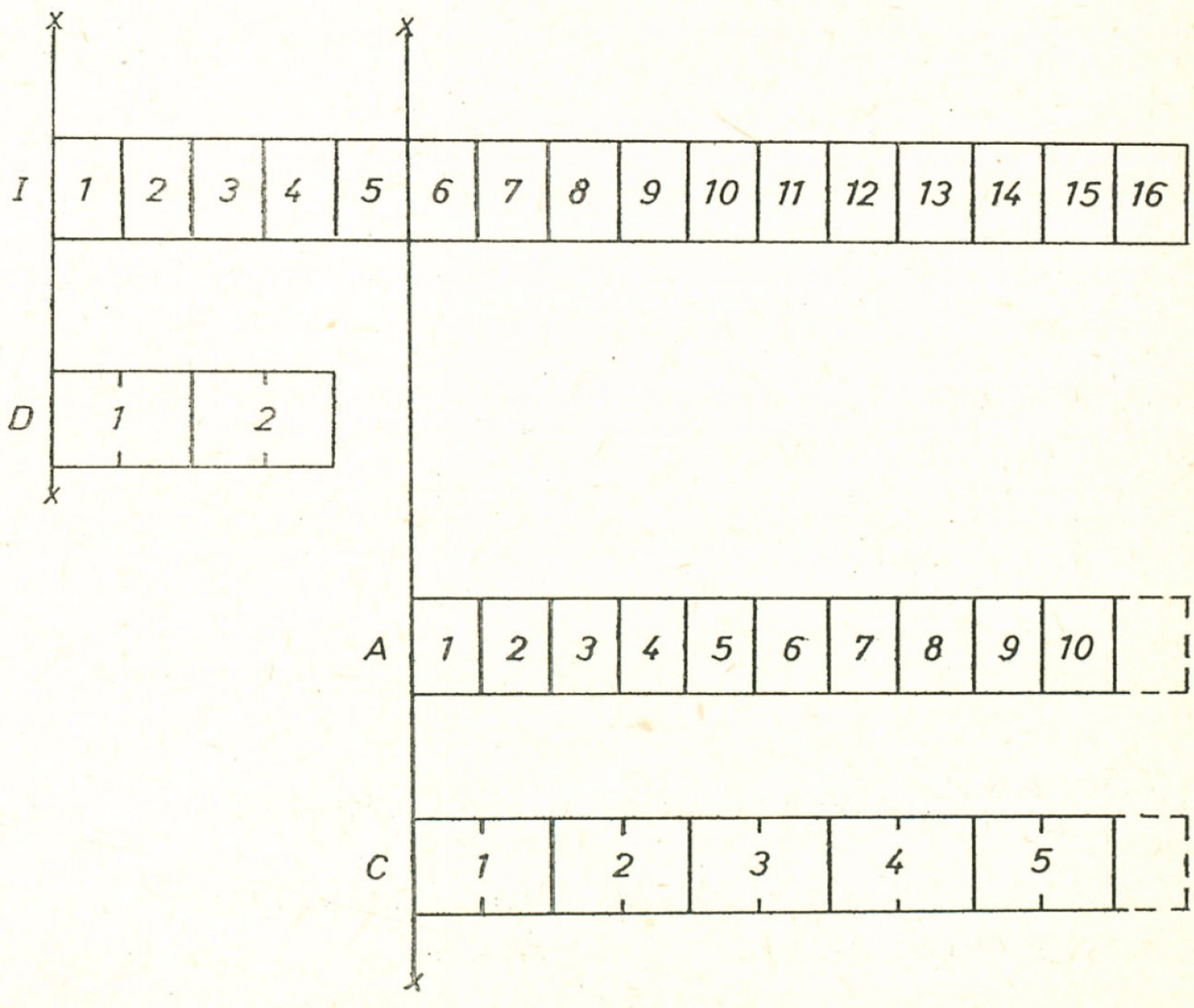
Az IBM FORTRAN-ban az EQUIVALENCE utasítás nem tér el a szabványtól.

Különböző típusú és hosszúságú változókat ekvivalenciába hozhatunk egymással, azonban mindig ügyelni kell arra, hogy a változók vagy tömbelemek címe hosszuknak megfelelően szóhatáron vagy duplaszóhatáron kezdődjön.

Példa:

```
INTEGER I (16)
REAL A (1Ø)
DOUBLE PRECISION C(5), D(2)
EQUIVALENCE (I(6), A(1), C(1)), (I(1), D(1))
.
.
.
```

Itt a C és D tömbök 8 bájt hosszúságúak, tehát duplaszóhatáron kell kezdődniük. Az első ekvivalenciacsoport szerint az I(6) elemnek és C(1)-nek azonos címen kell lennie, a második ekvivalenciacsoport pedig előírja, hogy az I és D tömbök is azonos címen kezdődjenek (duplaszóhatáron) (22. ábra). Ez] az EQUIVALENCE utasítás hibajelzést fog okozni. A fordítóprogram csak informatív hibajelzést ad, azonban a szóhatárok megsértése futás közben programmegszakítást eredményez.

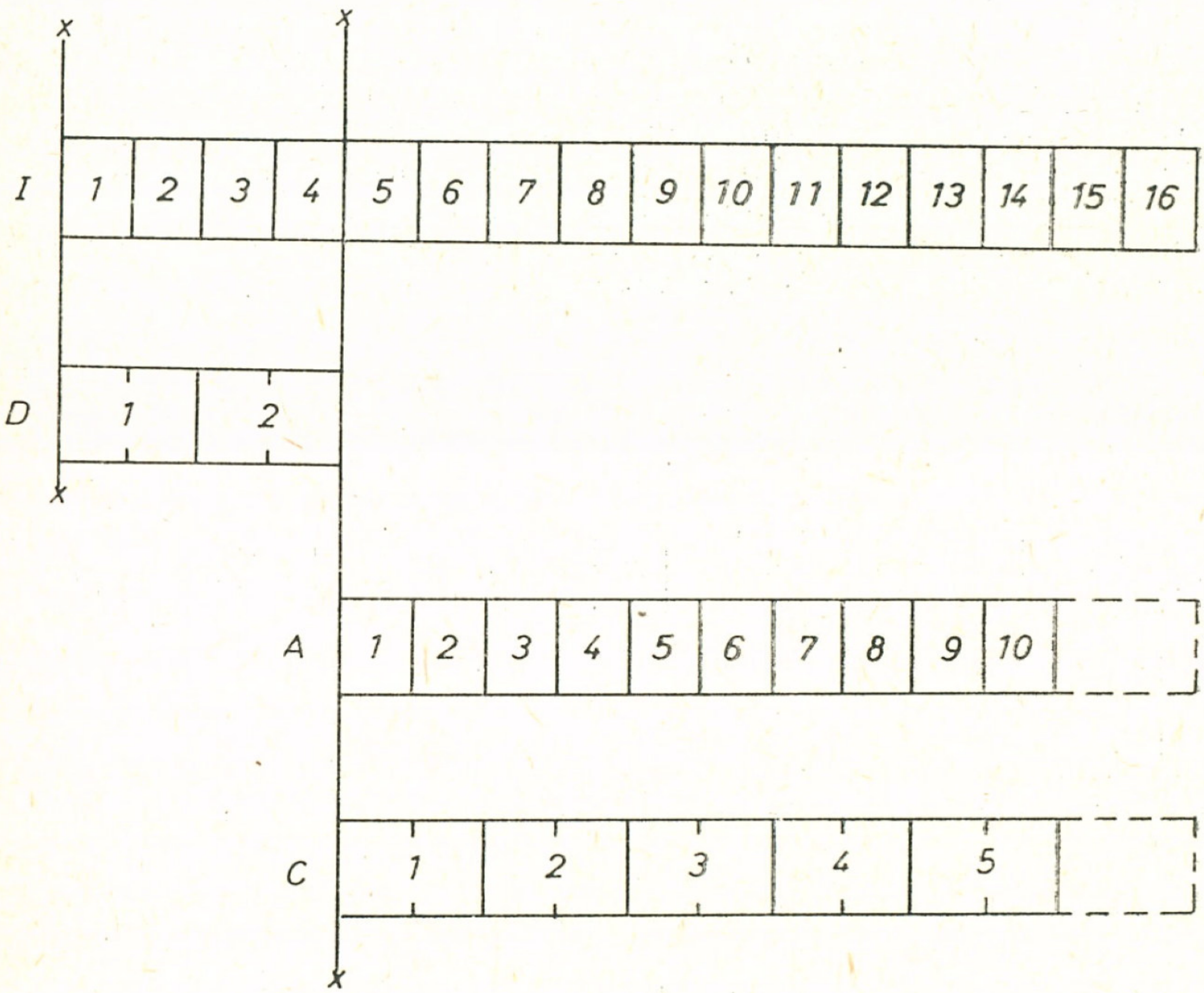


22. ábra

Az előző példa helyesen:

```
INTEGER I(16)
REAL A(1Ø)
DOUBLE PRECISION C(5), D(2)
EQUIVALENCE (I(5), A(1), C(1)), (I(1), D(1))
.
.
.
```

A helyesen felírt ekvivalencia képét a 23. ábra szemlélteti.



23. ábra

5. Kezdőérték adás. A DATA utasítás

A DATA utasítást változók, tömbelemek és *tömbök* kezdőértékének beállítására használjuk. Alakja:

$$\text{DATA } k_1/d_1/,\ k_2/d_2/,\ \ldots,\ k_n/d_n/$$

ahol k_i változónevekből, tömbelemekből vagy *tömbazonosítókból álló lista; d_i* egész, valós, komplex, hexadecimális és logikai konstansokból vagy szövegkonstansokból (Hollerith) álló lista. Bármelyik konstanst megelőzhet egy *i*∗ ismétlési tényező. Az utasításban szereplő k_i és d_i listák elemeinek száma meg kell, hogy egyezzen. Ha a k_i lista tömbnevet tartalmazott, akkor a konstansok listájában, az ismétlési tényezőt is figyelembe véve, annyi elemnek kell szerepelnie, amennyi elemet deklaráltunk a tömbnek.

Az egész, valós, komplex konstansoknak ugyanolyan típusúaknak kell lenniük, mint annak a változónak vagy tömbnek, amelynek a kezdőértéket adtuk, de bármilyen típusú változónak vagy tömbnek adhatunk hexadecimális vagy szövegkonstans kezdőértéket is.

A DATA utasítást bárhol elhelyezhetjük a programban, de nem előzhet meg olyan deklarációs utasítást, amelyben ugyanarra a változóra vagy tömbre hivatkozunk, mint amelynek kezdőértéket adtunk. Állhat azonban más változókra vonatkozó deklarációs utasítások előtt.

Egy változónak vagy tömbelemnek csak egyszer adhatunk kezdeti értéket.

Blank COMMON utasításban szereplő változónak vagy tömbelemnek nem adhatunk kezdőértéket. COMMON utasításban felsorolt változók vagy tömbelemek csak BLOCK DATA szegmensben kaphatnak kezdőértéket (l. a függelék Ötödik fejezetének 6. alfejezetét).

Példák

```
DIMENSION D(5,1Ø)
DATA A, B, C/5.Ø,6.1,7.3/, D, E/25∗1.Ø,25∗2.Ø,5.1/
.
.
.
```

Az A, B, C és E valós típusú változóknak rendre az 5.Ø, 6.1, 7.3, 5.1 értéket, a D tömb első 25 elemének 1.Ø, további 25 elemének 2.Ø értéket adtunk.

```
INTEGER A
DATA A/ZF79A/
```

Az A változó kezdeti értéke az F79A hexadecimális szám lesz.

a) Szövegkonstansok

A szövegkonstans betűkből, számokból és/vagy speciális karakterekből álló karakterlánc (string). Kétféle módon jelölhetjük:

— *a karakterláncot idézőjelek közé zárjuk;*

— a karakterlánc egy *d*H specifikációval kezdődik, ahol

d a láncban levő karakterek számát jelöli.

A szövegkonstansban legfeljebb 255 karakter lehet.

Ha a szövegkonstanst idézőjelek között adjuk meg, az idézőjelet a láncon belül két idézőjellel kell jelölni. Ha a láncot *d*H specifikációval kezdjük, az idézőjel önmagát jelenti.

Szövegkonstanst csak szubrutinok argumentumaként, DATA utasításban, explicit típusdeklarációban kezdőértékként, valamint a FORMAT és a PAUSE utasításokban adhatunk meg.

Példa szövegkonstansokra

'ALFA' vagy 4HALFA
'A' 'FORTRAN' 'NYELV' vagy 17HA 'FORTRAN' NYELV

A következő példákban változók és tömbök kezdőértékét szövegkonstanssal adjuk meg.

```
DIMENSION A(3)
DATA A/'ABCD', 'EFGH', 'IJKL'/
```

Az utasítás hatására

A(1) értéke ABCD,
A(2) értéke EFGH,
A(3) értéke IJKL lesz.

```
DIMENSION A(3)
DATA A/'ABCDEFGHIJKL'/
```

Az utasítás hatására a tömb elemeinek értéke ugyanaz lesz, mint az előző példában, de a fordítóprogram informatív hibaüzenetet küld, mivel a tömb elemeinek száma és a konstanslista elemeinek száma nem egyezik meg.

```
   | DIMENSION D(2), A(4)
1Ø | DATA D, J/'ABCDEFGH', 'XYZW'/
2Ø | DATA A/14HABCDEFGHIJKLMN/
```

A 1Ø-es címkéjű utasítás hatására a D tömb elemeinek tartalma

D(1) = ABCD
D(2) = XYZW

lesz, míg a J változó nem kap értéket.

A 2Ø-as címkéjű utasítás hatására az A tömb elemeinek tartalma a következő lesz:

A(1) = ABCD
A(2) = EFGH
A(3) = IJKL
A(4) = MN▽▽

Az A(4) elem jobbról szóközkarakterekkel egészül ki.

6. Utasításfüggvények

Az utasításfüggvény típusát automatikus típusdeklarációval, IMPLICIT utasítással vagy explicit típusdeklarációval lehet megadni. A következőkben az utasításfüggvény egy speciális használatára mutatunk példát.

Ha egy programban az egyszeres pontosságú számolásról kétszeres pontosságra kívánunk áttérni, minden matematikai függvényt kétszeres pontosságú változatára kell kicserélnünk. Ezt megtehetjük egyszerűen úgy, hogy a program vagy szubrutin elejére illesztett utasításfüggvényekkel megváltoztatjuk a nevek értelmezését.

```
IMPLICIT REAL*8 (A—H,O—Z)
.
.
.
SQRT(X) = DSQRT(X)
SIN(X) = DSIN(X)
.
.
.
A = SQRT(B)
C = SIN(F)
.
.
.
```

Az IMPLICIT utasítással felülbíráltuk az automatikus típus-deklarációt, ennek hatására az SQRT függvénynév kétszeres pontosságú függvényt jelöl, aminek definíciója a DSQRT függvény hívását írja elő. Ha például kétszeres pontosságú komplex mennyiségekkel kívánunk számolni és a CSQRT függvényt az előző példa mintájára újra definiáljuk, akkor a CSQRT függvény típusát is kétszeres pontosságú komplex típusúnak kell deklarálnunk:

```
IMPLICIT REAL*8(A,B,D—H,O—Z),COMPLEX*16(C,K)
.
.
.
SQRT(X) = DSQRT(X)
CSQRT(K) = CDSQRT(K)
.
.
.
A = SQRT(B)
C = CSQRT(K)
.
.
.
```

Mivel a C és K betűvel kezdődő neveket kétszeres pontosságú komplex mennyiségeknek deklaráltuk, a CSQRT függvény hívásakor a CDSQRT kétszeres pontosságú függvény fog aktivizálódni.

Az utasításfüggvények ilyen módon való felhasználásával elkerülhetjük azt a hibát, hogy a program egyes helyein megmaradnak az egyszeres pontosságú függvényhívások.

Ötödik fejezet:

TÖBB SZEGMENSBŐL ÁLLÓ FORTRAN PROGRAMOK

(A könyv és a függelék fejezetszámozásának párhuzamosságát mindvégig szeretnénk megőrizni, így az ötödik fejezetet a 2. alfejezettel kezdjük. A könyvben az 1. alfejezet áttekintő bevezetés volt.)

2. Szegmensnyitó és -záró utasítások

Az IBM FORTRAN-ban szegmensnyitó utasítások a következők:

```
SUBROUTINE
FUNCTION
BLOCK DATA
```

A főprogramban nincs szegmensnyitó utasítás. A szegmenszáró utasítás:

```
END
```

3. Szubrutinok és függvények aktivizálása. Aktuális és formális paraméterek

a) SUBROUTINE szegmensek. A CALL és a RETURN utasítás

A SUBROUTINE utasítás alakja:

SUBROUTINE *név* $(a_1, a_2, \ldots, a_n)$

ahol *név* a szubrutin neve; a_i egymástól különböző, nem kötelező formális paraméterek. Ha a SUBROUTINE utasításnak nincs formális paramétere, akkor a zárójelek is elhagyhatók. A for-

mális paraméter lehet változó- vagy tömbnév, amely *megadható két „/" jel közé zárva is* (a két „/" jel közé zárt paraméter név szerinti paraméterátvételt jelöl); más szubrutin vagy függvény neve; „*" *karakter*, amely egy, a hívóprogramba való visszatérési pontot (egy utasításcímkét) jelöl.

A SUBROUTINE utasításnak a programszegmens első utasításának kell lennie. A szubrutin tartalmazhat bármilyen FORTRAN utasítást, kivéve egy másik SUBROUTINE, FUNCTION vagy BLOCK DATA utasítást. Ha IMPLICIT utasítást használunk a szubrutinban, akkor annak közvetlenül a SUBROUTINE utasítás után kell állnia. A szubrutin nevének azonosítója nem fordulhat elő a SUBROUTINE utasításon kívül más utasításban.

A formális paraméterek nem fordulhatnak elő COMMON, EQUIVALENCE, DATA és NAMELIST utasításban. Példák a SUBROUTINE utasításra:

```
SUBROUTINE ALFA
SUBROUTINE BETA (I,*, J,*)
SUBROUTINE GAMMA (/K/, B, /X/)
```

Az ALFA nevű szubrutinnak nincs formális paramétere. A BETA nevű szubrutin formális paraméterei között két „*" karakter szerepel. Ezek nem változó-, ill. tömbnevek, ezért a szubrutin formális paramétereinek listáján kétszer is előfordulhatnak. A „*" karakterrel azt jelöljük, hogy a paraméter helyébe egy, a hívószegmensben szereplő utasításcímkét kell írni. (Ennek részletes magyarázatára még visszatérünk.) A GAMMA szubrutinban a K és X formális paraméterek név szerinti paraméterátvételt írnak elő (l. a következőkben).

A szubrutinok aktivizálására (hívására) a CALL utasítás szolgál. Alakja:

CALL *név* $(a_1, a_2, \ldots, a_n)$

ahol *név* a szubrutin neve vagy egy ENTRY utasításban megadott név (l. az ENTRY utasítást); a_i az aktuális paramétereket jelöli. Aktuális paraméter lehet változó, tömbelem, tömbnév,

aritmetikai, logikai vagy Hollerith-konstans, aritmetikai vagy logikai kifejezés, szubrutin- vagy függvénynév, és a hívó programszegmens egy végrehajtható utasításának címkéje; ilyenkor a címke megadási formája *&címke* alakú. Hexadecimális konstans nem lehet aktuális paraméter.

A CALL utasításban az aktuális paraméterek száma, sorrendje, típusa és hossza azonos kell, hogy legyen a formális paraméterek számával, sorrendjével, típusával, ill. *hosszával*. Ha aktuális paraméter szubrutin- vagy függvénynév, azt a CALL utasítást kiadó programszegmensben EXTERNAL-ként deklarálni kell.

Példák CALL *utasításra*

```
CALL ALFA
CALL BETA (J,&1Ø,L,&2Ø)
CALL GAMMA (M, A, B)
```

A BETA szubrutin hívásakor az &1Ø és &2Ø aktuális paraméterek a hívó programrész végrehajtható utasításainak címkéi. A GAMMA szubrutinban az M és B aktuális paraméterek név szerint adódnak át, mivel GAMMA deklarációjában az első és harmadik formális paramétert „/" jelek közé zártuk.

Minden szubrutin szegmensben legalább egy RETURN utasításnak kell lennie.

A RETURN utasítás alakja:

RETURN
RETURN *i*

ahol *i* egész konstans vagy 4 bájt hosszúságú egész változó lehet. *i* a SUBROUTINE utasítás *i*-edik „*"-gal jelölt formális paraméterét jelöli ki.

A főprogramban kiadott RETURN utasítás hatása a STOP utasítás hatásával azonos.

A szabványnak megfelelő RETURN utasítás a szegmens logikai végét jelenti, végrehajtása után a hívószegmensben a CALL után következő utasítás fog végrehajtódni.

A RETURN *i utasítás lehetőséget nyújt arra, hogy a hívószegmens bármelyik címkézett végrehajtható utasítására visszatérhessünk.*

A RETURN *i* utasítás csak szubrutin szegmensben adható ki. Ha a szubrutinban deklarált „*" paraméterek számát *n*-nel jelöljük, *i* értékének az $1 \leqq i \leqq n$ tartományba kell esnie.

A következőkben a „*" formális paraméter és a RETURN *i* utasítás, valamint a név szerint történő paraméterátvétel lehetőségeit mutatjuk be.

a1)* **„*" formális paraméter és a RETURN *i* használata**

Tekintsük a következő szubrutint:

```
      SUBROUTINE SUB (Z,C,*,*,W,*)
      IF(Z) 1,2,2
    1 W = 4.Ø
      RETURN 1
C     VISSZATERES AZ 1. "*" ARGUMENTUMNAK
C        MEGFELELO CIMKERE
    2 IF(C) 3,4,5
    3 W = 5.Ø
      RETURN
C     VISSZATERES A CALL UTASITAST KOVETO
C        UTASITASRA
    4 W = Ø.Ø
      RETURN 3
C     VISSZATERES A 3. "*" ARGUMENTUMNAK
C        MEGFELELO CIMKERE
    5 W = C
      RETURN 2
C     VISSZATERES A 2. "*" ARGUMENTUMNAK
C        MEGFELELO CIMKERE
      END
```

A szubrutin a Z és C bemenő paraméterek értékétől függően a W kimenő paraméternek különböző értéket ad és a hívószegmens más és más helyére tér vissza.

Ha a főprogramból vagy más szegmensből a

```
CALL SUB (A,B,&1Ø,&2Ø,C,&3Ø)
```

utasítással hívjuk a SUB szubrutint és A < Ø, akkor W értéke 4.Ø lesz és a hívószegmens a 1Ø-es címkéjű utasításon fog foly-

tatódni. Ha A ≧ Ø és C < Ø, W értéke 5.Ø lesz, és a CALL utasítás után következő utasításra adódik vissza a vezérlés.

Ha a szubrutint a

```
CALL SUB (2.Ø, Ø.Ø, &1ØØ, &2, W, &2Ø)
```

utasítással hívtuk meg, W értéke Ø.Ø lesz, és a hívószegmens a 2Ø-as címkéjű utasítással folytatódik.

Hogy a „*" formális paraméter és a RETURN *i* utasítás használatát jobban megértsük, írjuk át a SUB szubrutint a következőképpen:

```
      SUBROUTINE SUB1(Z,C,W,I)
      IF(Z) 1,2,2
1     W = 4.Ø
      I = 1
      RETURN
2     IF(C) 3,4,5
3     W = 5.Ø
      I = Ø
      RETURN
4     W = Ø.Ø
      I = 3
      RETURN
5     W = C
      I = 2
      RETURN
      END
```

A hívószegmens most így alakul:

```
.
.
.
CALL SUB1(A,B,C,I)
GO TO (1Ø,2Ø,3Ø),I
.
.
.
```

Itt kihasználtuk az IBM FORTRAN-nak azt a tulajdonságát, hogy a kiszámított GO TO utasítás végrehajtásakor, ha I értéke kívül esik az $1 \leqq I \leqq n$ tartományon (ahol n a felsorolt utasításcímkék száma) a GO TO után következő utasításra adódik át a vezérlés.

Látható, hogy a SUB és a SUB1 szubrutin, ha hívása után közvetlenül a kiszámított GO TO utasítást is kiadjuk, azonos hatású.

Itt szeretnénk felhívni a figyelmet arra, hogy a „*" paraméter használata megnehezítheti a hibakeresést.

*a2**) Név szerint történő paraméterátvétel

A „/" jelek közé zárt formális paramétereket a szubrutin és a függvényszegmens név szerint veszi át. Az ilyen paraméterek számára a programszegmens nem foglal le tárterületet, a számítások elvégzéséhez a hívószegmens tárterületét használja. Ennek hatására a paraméter aktuális értéke egyszerre változik meg a hívó és a hívott programszegmensben. Ezzel ellentétben, az érték szerint átvett paraméterek számára a hívott programszegmensben tárterület jelölődik ki. Amikor a hívott programszegmensbe megtörténik a belépés, az aktuális paraméter értéke is betöltődik a programszegmens területén kijelölt helyre. Miután a szubrutin (vagy függvény) befejeződött, az aktuális paraméter végső értéke adódik vissza a hívó programszegmensnek.

A „/" jelek közé zárt formális paramétereknek a közvetlen hozzáférésű adatátvitelek esetében van jelentős szerepe, részletesebb magyarázatára ott még visszatérünk.

b) FUNCTION szegmensek és alkalmazásaik

A FUNCTION utasítás alakja:

típus FUNCTION *név***s*$(a_1, a_2, \ldots, a_n)$

ahol *típus* az INTEGER, REAL, DOUBLE PRECISION, COMPLEX vagy LOGICAL FORTRAN alapszavak egyike. Nem kötelező kiírni; ha elhagyjuk, az automatikus típusdeklaráció határozza meg a függvény típusát; *név* a függvény neve; *s* valamilyen megengedett hosszspecifikációt jelöl (csak akkor ad-

ható meg, ha a típust is megadtuk, DOUBLE PRECISION típus esetén nem használható); a_i egymástól különböző formális paraméterek. Legalább egy formális paraméter megadása kötelező. A formális paraméter lehet változó- vagy tömbnév, amely két „/" jel közé zárva is megadható; ill. más függvény vagy szubrutin neve.

Egy függvény típusa és hossza megadható automatikus típusdeklarációval (ekkor szabványos hosszúságú lesz), explicit specifikációval a FUNCTION utasításban, *és a* FUNCTION *szegmensen belül* IMPLICIT *utasítással vagy explicit típusdeklarációval.* Ha a függvény típusa nem az automatikus típusdeklarációnak felel meg, a függvény típusát deklarálni kell a hívóprogramban is.

A FUNCTION utasításnak a programszegmens első utasításának kell lennie. A függvény tartalmazhat bármilyen FORTRAN utasítást, kivéve SUBROUTINE, másik FUNCTION vagy BLOCK DATA utasítást. Ha a függvényszegmensben az IMPLICIT utasítást használjuk, akkor annak közvetlenül a FUNCTION utasítás után kell állnia.

A függvény nevének (vagy az ENTRY névnek) a szegmensen belül értéket kell adni.

Egy függvény aktuális argumentuma lehet Hollerith-konstans. Az aktuális argumentumként átadott Hollerith-konstans az őt elhatároló idézőjelek nélkül, vagy a *d*H specifikáció nélkül adódik át.

Példák FUNCTION *szegmensekre*

```
FUNCTION ALFA (X,Y)
.
.
.
ALFA = X+Y+15.Ø*X**2
RETURN
END
```

Az ALFA nevű függvény valós típusú, 4 bájt hosszúságú lesz.

```
COMPLEX FUNCTION KOBGYO*16(Z)
COMPLEX*16 Z
.
.
.
KOBGYO = ...
RETURN
END
```

A KOBGYO nevű függvényt komplex típusú, 16 bájt hosszúságú függvénynek definiáltuk.

Hívása:

```
.
.
.
COMPLEX*16   KOBGYO, X, Y
.
.
.
Y = KOBGYO(X)
.
.
.
```

Mivel a KOBGYO az automatikus típusdeklaráció szerint egész típusú lenne, a hívószegmensben 16 bájt hosszúságú, komplex típusúnak kell deklarálni.

```
FUNCTION NUMBER(K)
INTEGER*2 NUMBER, K
.
.
.
NUMBER = ...
.
.
.
RETURN
END
```

Ebben a példában kihasználtuk, hogy a függvény típusa és hossza a FUNCTION szegmensen belül explicit típusdeklarációval is megadható. Hívásakor a hívószegmensben is deklarálni kell a típust és hosszt.

```
      .
      .
      .
      INTEGER*2 NUMBER, M, N, J, K
      .
      .
      .
      M = NUMBER(N)+J*K
```

A következő példában IMPLICIT utasítással adjuk meg a függvény típusát és hosszát:

```
      FUNCTION BETA (X,Y)
      IMPLICIT REAL*8 (A—H,O—Z)
      .
      .
      .
      BETA = ...
      RETURN
      END
```

Hívása

```
      IMPLICIT REAL*8 (A—H,O—Z)
      .
      .
      .
      W = BETA (X,Y)
      .
      .
      .
```

c)* Az ENTRY utasítás

Egy szubrutin vagy függvényszegmens végrehajtása az első végrehajtható utasítással kezdődik, másképpen fogalmazva a

szubrutin CALL utasítással való aktivizálása, ill. egy függvényre való hivatkozás hatására a szubrutin vagy függvény első végrehajtható utasítására történik a belépés. Az IBM FORTRAN-ban egy szubrutin vagy függvényszegmensnek több belépési pontja is lehet. Ezeket a belépési pontokat az ENTRY utasítással kell kijelölni.

Az ENTRY utasítás alakja:

ENTRY *név* $(a_1, a_2, \ldots, a_n)$

ahol *név* a belépési pont neve; a_i formális paraméter, amely a CALL utasításban vagy a függvényhivatkozásban egy aktuális paraméternek felel meg. A formális paramétereket két „/" jel közé is zárhatjuk. A SUBROUTINE utasításban megadott ENTRY utasítás állhat formális paraméter nélkül is, ilyenkor a zárójelek is elmaradnak. FUNCTION szegmensben deklarált ENTRY utasításnak legalább egy paraméterének kell lennie.

Példák

```
SUBROUTINE ALFA (X,Y)
.
.
.
ENTRY BETA (X)
.
.
.
RETURN
END
```

```
FUNCTION GAMMA (A,B)
.
.
.
ENTRY DELTA (A,B)
.
.
.
RETURN
END
```

Egy szubrutinban definiált ENTRY belépési pontot CALL utasítással, függvényben definiált belépési pontot függvényhivatkozással aktivizálhatunk. Például

```
.
.
.
CALL ALFA (Z,W)
.
.
.
CALL BETA (V)
.
.
.
C = GAMMA (X,Y)
.
.
.
B= DELTA (P,Q)
.
.
.
```

Az ENTRY nemvégrehajtható utasítás. Az ENTRY utasítás aktivizálásakor a szubrutin vagy függvényszegmensbe az ENTRY utasítást követő első végrehajtható utasításra történik a belépés.

DO cikluson belül belépési pont nem definiálható.

Az ENTRY utasítás formális paramétereinek száma, típusa és sorrendje nem kell, hogy megegyezzen a SUBROUTINE vagy FUNCTION utasításban deklarált, vagy ugyanabban a szegmensben definiált más ENTRY utasítás formális paraméterekéivel. Azonban az ENTRY utasítást aktivizáló CALL utasításban vagy függvényhivatkozásban az aktuális paramétereknek számban, típusban és sorrendben meg kell egyezniük az ENTRY utasítás formális paramétereivel.

A FUNCTION szegmensben definiált ENTRY utasítás nevének típusát és hosszát az automatikus típusdeklaráció, az

IMPLICIT utasítás, explicit típusdeklaráció vagy a FUNCTION utasítás típusa határozza meg. A függvénynév és az ENTRY név típusa lehet különböző, de a tárban ugyanazt a helyet foglalják le. A hosszt mindig a FUNCTION utasításban használt név hossza határozza meg.

Függvényszegmensben vagy a függvény nevének, vagy az ENTRY utasításban használt névnek egy értékadó utasítás bal oldalán kell állnia.

Függvényből való visszatéréskor a függvény értéke az utoljára végrehajtott értékadó utasításban szereplő függvénynévnek vagy ENTRY névnek adott érték lesz, függetlenül attól, hogy hogyan hivatkozhatunk a függvényre. Ha a függvénynév és az ENTRY név különböző típusú, a függvény értéke határozatlan.

A következőkben példákon keresztül mutatjuk be az ENTRY utasítás használatát.

```
   SUBROUTINE SUB (X,Y,P)
   IF(X) 11, 12, 12
11 X =—X
12 Y = Y+X
   ENTRY SUB1 (Y,P)
13 P = 5.Ø*Y**2
   RETURN
   END
```

A szubrutinnak két belépési pontja van: SUB és SUB1. Ez tulajdonképpen két szubrutin, amelyet egy szegmensben írtunk. A

```
CALL SUB (A,B,C)
```

hívásakor C-ben $5.Ø*(B+|A|)^2$ eredményt,

```
CALL SUB1 (B,C)
```

hívásakor C-ben $5.Ø*B^2$ eredményt kapunk. Ha azonban a 13-as címkéjű utasítást módosítjuk:

```
13 P = 5.Ø*Y**2+X
```

már nem beszélhetünk két szubrutinról, mivel a hívások eredménye X aktuális értékétől is függ. Ekkor a

```
CALL SUB1 (B,C)
```

utasítás kiadásának csak akkor van értelme, azaz C-ben csak akkor kapjuk a kívánt eredményt, ha előzőleg már a

```
CALL SUB (A, B, C)
```

utasítással X-nek értéket adtunk, és SUB1 hívásakor ezzel az értékkel történik C kiszámítása.

Tekintsük a következő FUNCTION szegmenst:

```
  FUNCTION FUNC1 (X1,X2,X3,X4)
  FUNC1 = X1*X2+X3*X4
1 RETURN
  ENTRY FUNC2 (X4)
  IF(X4) 2,2,5
2 FUNC2 = X1*X4+X3
  RETURN
5 FUNC2 = 10.0
  RETURN
  END
```

Az előző példához hasonlóan most is először a FUNC1 függvényt kell hívni, és csak azután van értelme a FUNC2 belépési pontot aktivizálni. Hívása például:

```
.
.
.
F = FUNC1 (A,B,C,D)
.
.
.
F1 = FUNC2(D)*A**2
.
.
.
```

A függvényszegmens megírásánál ügyelnünk kell arra, hogy a függvényérték a legutolsó értékadó utasításban szereplő függvénynévnek vagy ENTRY névnek adott érték lesz, függetlenül attól, hogy hogyan aktivizáltuk a függvényt. Ha az előbbi példában az 1-es címkéjű RETURN utasítást nem írjuk le, az

```
F = FUNC1(A,B,C,D)
```

és az

```
F1 = FUNC2(D)
```

függvényhívások eredménye, feltéve, hogy FUNC2-t megelőzően FUNC1-et is hívtuk, ugyanaz az érték lesz: A*D+C ha D < Ø és 1Ø.Ø ha D > Ø.

Erre különösen vigyázni kell, ha a függvénynév és az ENTRY név nem azonos típusú. Tekintsük például a következő függvényt:

```
  | FUNCTION W(X)
1 | W = X**2
  | ENTRY K(X)
  | K = X**3
  | RETURN
  | END
```

A W függvény aktivizálása rossz eredményt ad, mivel a legutolsó értékadó utasításban az ENTRY név szerepel és az ENTRY név típusa nem egyezik meg a függvénynév típusával. Ha azonban az 1-es címkéjű utasítás és az ENTRY utasítás közé egy RETURN utasítást írunk, a

```
V = W(2.Ø)
```

függvényhívás a helyes eredményt szolgáltatja. Vegyük észre, hogy ebben az esetben két függvényről van szó, amelyet egy szegmensben írtunk!

A név szerint átadott paramétereknek az ENTRY utasítás paraméterlistájában is szerepelniük kell.

Ha egy dinamikus indexhatárokkal deklarált tömb egy ENTRY utasítás paramétere, akkor a tömb összes dimenzió-

jára vonatkozó információnak is szerepelnie kell a paraméterlistában.

Bonyolult programszegmensek írásánál elegáns megoldásként alkalmazhatjuk az ENTRY utasítást, azonban fel kell hívnunk a figyelmet arra, hogy használata sok hibalehetőséget rejt magában. Kevés gyakorlattal rendelkező programozó számára nem javasoljuk használatát.

4. Dinamikus indexhatárok

SUBROUTINE vagy FUNCTION szegmensekben formális paraméterként deklarált tömbök méretét változóként is definiálhatjuk. A tömböt deklaráló utasításban az indexhatárokat 4 bájt hosszúságú egész változóknak kell megadni. A tömbök indexhatárait jelölő változóknak vagy maguknak is formális paramétereknek kell lenniük, vagy COMMON *utasításban kell szerepelniük*. A tömbök abszolút méretét a hívószegmensben vagy egy magasabb szinten levő hívószegmensben kell deklarálni. A méretet definiáló változók aktuális értékének olyannak kell lenniük, hogy a formális paramétertömb mérete kisebb vagy egyenlő legyen az aktuális tömb méretével. A tömbök dinamikus indexhatárait jelölő változók értékét a szegmensben nem szabad megváltoztatni.

Példák

Az indexhatárt jelölő N egész változó maga is formális paraméter:

```
SUBROUTINE SUB (A,N)
DIMENSION A(N)
.
.
.
```

Az indexhatárt jelölő N egész változó COMMON utasításban szerepel

```
SUBROUTINE SUB1 (A)
COMMON /X/ N
DIMENSION A(N)
```

Választhatjuk a formális paramétertömb dimenzióját konstansnak is, mégpedig úgy, hogy mérete kisebb legyen az aktuális tömb méreténél.

```
      SUBROUTINE SUB2 (A,N)
      DIMENSION A(1)
      .
      .
      .
      DO 1Ø I = 1,N
1Ø    A(I) = ...
      .
      .
      .
      RETURN
      END
```

Ez nem mond ellent annak a szabálynak, hogy az aktuális paramétertömb elemeinek száma legalább akkora kell, hogy legyen, mint a formális paramétertömb elemeinek száma. Az IBM FORTRAN-ban a formális paramétertömb elemeinek számát gyakran szokták 1-nek választani. A SUB2 aktivizálása lehet például:

```
      CALL SUB2(B,1ØØ)
```

5. Közös adatmezők. A COMMON utasítás

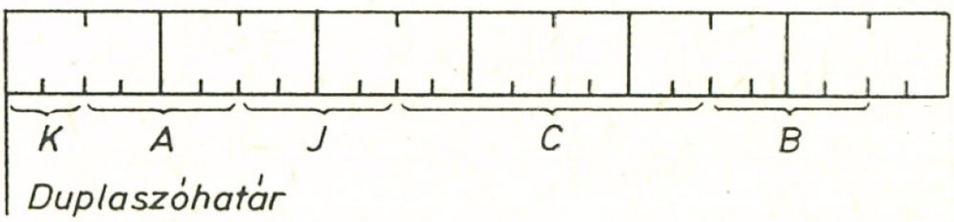
A COMMON utasításban a változókat és tömböket hosszuk csökkenő sorrendjében kell deklarálni; először a COMPLEX 16 bájt hosszúságúakat, azután a COMPLEX vagy REAL 8 bájt hosszúságúakat stb. Legutoljára az 1 bájt hosszúságú LOGICAL típusú változókat írjuk.

A COMMON blokkba a változók a COMMON utasításban való felsorolási rendjük szerint kerülnek, ilyen sorrendben foglalják le a tárterületet. A COMMON blokk mindig duplaszóhatáron kezdődik. Ha nem vigyázunk a COMMON blokk elrendezésére, előfordulhat, hogy egy változó címe nem megfelelő szóhatáron kezdődik. Ilyenkor a fordítóprogram informatív hibaüzenetet küld, a program futása közben pedig programmegszakítás következik be.

Példa

```
COMPLEX C
INTEGER*2 K
COMMON /BAB/ K, A, J, C, B
.
.
.
```

Itt K hossza 2 bájt, C hossza 8 bájt, A, B, J pedig 4 bájt hosszúságú. A BAB nevű COMMON blokk tárbeli elrendezését a 24. ábra szemlélteti.



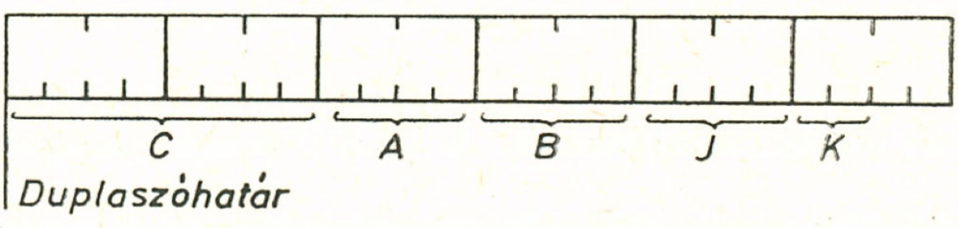
24. ábra

Látható, hogy K változó az egyetlen, amely a megfelelő szóhatáron kezdődik, az A, J, C, B változók használata programmegszakítást fog eredményezni.

A COMMON blokk helyes felírása:

```
COMPLEX*C
INTEGER*2 K
COMMON /BAB/C,A,B,J,K
```

A helyes tárbeli elrendezést a 25. ábra szemlélteti.



25. ábra

6. Közös adatmezőkben szereplő változók kezdőértékének beállítása. BLOCK DATA szegmensek

A COMMON utasításban szereplő változók kezdőértékét csak BLOCK DATA szegmensben lehet beállítani.

A BLOCK DATA szegmens csak a BLOCK DATA, DATA, COMMON, DIMENSION, EQUIVALENCE utasításokat, IMPLICIT és explicit típusdeklaráló utasításokat tartalmazhat. Első utasítása a

```
BLOCK DATA
```

utasítás kell, hogy legyen. Ezután, ha van, az IMPLICIT utasításnak kell következnie.

Annak a COMMON utasításnak, amelyben szereplő változóknak kezdeti értéket adunk, mindig meg kell előznie a kezdeti értéket beállító utasítást. *Kezdeti értéket explicit típusdeklaráló utasítással is adhatunk.*

Példa

```
BLOCK DATA
COMMON /ELSO/ A,B,C
REAL*8 A/3.25484578DØ/,B/2.ØDØ/
COMPLEX C/(2.4,3.769)/
COMMON/MASODI/ P,Q,R(5)
INTEGER P/15/,Q/3/
DATA R/7.25,4.16,3*2.Ø/
END
```

7. Szegmensnevek alkalmazása aktuális és formális paraméterként. Az EXTERNAL utasítás

Az IBM FORTRAN-ban az EXTERNAL utasítás nem tér el a szabványtól.

Az EXTERNAL utasítás kapcsán felhívjuk a figyelmet a külső és belső standard függvények közötti különbségre. A belső standard függvényeket a fordítóprogram felismeri és a hívás helyén a programba beépíti. A külső standard függvényeket a fordítóprogram ugyanúgy kezeli, mint bármilyen

más szegmenst. (A fordítóprogram által lefordított szegmenseket a szerkesztőprogram szerkeszti össze egy programmá. A külső standard függvényeket a szerkesztőprogram egy könyvtárból olvassa be és hozzászerkeszti a programozó által írt szegmensekhez.)

Ha egy belső standard függvény neve EXTERNAL utasításban szerepel, a fordítóprogram nem tekinti belső standard függvénynek, hanem egy, a programozó által írt szegmensnek.

A következő példában az ABS belső standard függvényt EXTERNAL-ként definiáljuk.

```
EXTERNAL ABS
.
.
.
CALL SUB (ABS,X,Y)
.
.
.
END
SUBROUTINE SUB (F,X,Y)
Y = F(X)
RETURN
END
```

A fordítóprogram ebben az esetben az ABS belső standard függvényt nem ismeri fel; úgy kezeli mint egy, a programozó által írt FUNCTION szegmenst. Ezért kell még egy ABS nevű függvényszegmenst is írnunk:

```
FUNCTION ABS(X)
.
.
.
ABS = ...
RETURN
END
```

Külső standard függvény neve szerepelhet EXTERNAL utasításban anélkül, hogy a programozónak újra kellene írnia a függvényszegmenst. Például:

```
EXTERNAL SIN
.
.
.
CALL SUB (SIN,X,Y)
.
.
.
END
SUBROUTINE SUB (A,B,C)
.
.
.
C = A(B)
.
.
.
RETURN
END
```

Ha azonban a programozó ír egy SIN nevű függvényszegmenst, akkor ez a függvény fog beépülni a programba. Ezzel a belső standard függvényekhez hasonlóan átdefiniáltuk a külső standard függvényeket is.

Mind a belső, mind a külső standard függvények átdefiniálása azonban sok hibalehetőséget rejt magában, ezért célszerű elkerülni.

Hatodik fejezet

ADATÁTVITEL ÉS UTASÍTÁSAI

1. Periferiális egységek és külső tárolók

A periferiális egységekkel és a külső tárolókkal való adatcserében az átvitt adat mindig egy adatállományhoz (file) tartozik. A FORTRAN programban használt adatállományra mindig az *állomány hivatkozási számával* kell hivatkozni (a könyv hatodik fejezete a logikai perifériaszám elnevezést használja). Az adatállományok egy vagy több rekordból állnak.

Az IBM FORTRAN nyelvben soros hozzáférésű (szekvenciális) és *közvetlen hozzáférésű* (direkt) adatállományokat lehet feldolgozni. A soros szervezésű adatállományok fizikai hordozója lyukkártya, lyukszalag, mágnesszalag és mágneslemez. Közvetlen szervezésű adatállomány fizikai hordozója csak mágneslemez lehet.

Közvetlen hozzáférésű adatállományt a DEFINE FILE utasítással kell létrehozni, feldolgozását a megfelelő READ, WRITE és FIND utasítások támogatják.

A továbbiakban először a soros szervezésű adatállományokra vonatkozó adatátviteli utasításokat tárgyaljuk.

2. A READ és WRITE utasítások szintaxisa és szemantikája

A READ utasítás alakja:

READ (*a*,*b*,ERR = *c*, END = *d*) L

ahol *a* előjel nélküli egész szám, vagy egy 4 bájt hosszúságú egész típusú változó, az adatállomány hivatkozási száma*; *b* vagy a beolvasáshoz rendelt FORMAT utasítás címkéje, vagy egy tömbazonosító, amely esetben a tömb tartalmazza a

* A könyv hatodik fejezetében logikai perifériaszám néven szerepel.

FORMAT specifikációt. *b* kiírása nem kötelező, ha elhagyjuk, kötetlen formátumú* (unformatted) READ utasításról beszélünk. ERR = *c* nem kötelező paraméter. Ha kiírjuk, *c*-nek egy utasításcímkének kell lennie, amelyre az adatátvitel során esetleg bekövetkezett hiba észlelése után a vezérlés átadódik. END = *d* nem kötelező paraméter. Ha kiírjuk, *d*-nek egy utasításcímkének kell lennie, amelyre a vezérlés átadódik, ha az olvasás során az adatállomány vége jelzés érkezett (end-of-file). L az I/O listát jelöli.

Az ERR = *c* és END = *d* paraméterek sorrendje felcserélhető.

A READ *utasítás* ERR = *c* *paramétere lehetőséget nyújt a programozónak az adatátviteli hibák feldolgozására.* Ha a beolvasandó rekordban adathiba van, akkor az ERR = *c* paraméter hatására a *c* címkéjű utasításra adódik át a vezérlés, ahol a programozó dönthet a hibás rekord sorsáról. A hiba észlelése után következő READ utasítás ismét a hibás rekordot szolgáltatja.

Példa

```
      .
      .
      .
10    READ (10,20,ERR = 30) A
20    FORMAT (10F8.2)
      .
      .
      .
30    READ (10,100)AX
100   FORMAT (80A1)
      WRITE (6,101)AX
101   FORMAT ('▽HIBAS REKORD: ',80A1)
      .
      .
      .
      GO TO 10
      .
      .
      .
```

* A könyv hatodik fejezetében bináris adatátvitelként szerepel.

A 3Ø-as címkéjű READ utasítás a hibás rekordot szolgáltatja. A rekordot sornyomtatóra kiírjuk, esetleg valamilyen vizsgálatot is elvégezhetünk, majd visszatérünk az eredeti programrészbe. A következő READ utasítás a hibás rekord utáni rekordot fogja beolvasni.

Felhívjuk a figyelmet arra, hogy az ERR = *c* paraméter csak hibás rekordok kezelésére szolgál. A FORMAT utasításnak nem megfelelő adatok más jellegű hibára vezetnek, amelyet az ERR = *c* paraméterrel nem lehet kivédeni. Ha a READ utasítás ERR = *c* paraméterét elhagyjuk, hibás rekord olvasásakor a program végrehajtása hibajelzéssel fejeződik be.

A READ *utasítás* END = *d paraméterét jól lehet használni, ha a programozó előre nem tudja a beolvasandó adatok mennyiségét*. Például a rekordok szerkezetét ismerjük, de a rekordok számát nem.

```
        .
        .
        .
        I = Ø
   1Ø   READ (2Ø,3Ø,END = 2ØØ) A
C       A BEOLVASOTT ADATOK FELDOLGOZASA
        .
        .
        .
        I = I+1
        GO TO 1Ø
  2ØØ   WRITE (6,4Ø) I
C       KINYOMTATJUK A BEOLVASOTT REKORDOK
C       SZAMAT
        .
        .
        .
```

Ha a READ utasítás END = *d* paraméterét elhagyjuk, az utolsó rekord beolvasása után kiadott READ utasítás hatására a program végrehajtása hibajelzéssel fejeződik be.

A soros szervezésű adatállományokra vonatkozó WRITE utasítás megegyezik a könyv Hatodik fejezetében leírtakkal.

a)* További be- és kiviteli utasítások

Az itt tárgyalt három utasítás soros szervezésű adatállományok be- és kivitelére szolgál.

A READ *utasítás speciális formája a kártyaolvasót jelöli ki beviteli perifériának.*

Az utasítás alakja:

READ *b*,L

ahol *b* vagy egy FORMAT utasítás címkéje, vagy egy tömbazonosító, amely a FORMAT specifikációt tartalmazza; L az I/O listát jelöli.

Példa

```
   |  .
   |  .
   |  .
   |  READ 1Ø, A,B,C
1Ø |  FORMAT (3F7.3)
   |  .
   |  .
   |  .
```

A PUNCH *utasítás a kártyalyukasztót jelöli ki kiviteli perifériának.* Hatása azonos a WRITE (*u*,*b*)L utasítás hatásával, ahol *u* a kártyalyukasztóra írt adatállomány hivatkozási száma. Alakja:

PUNCH *b*,L

ahol *b* vagy egy FORMAT utasítás címkéje, vagy egy tömbazonosító, amely a FORMAT specifikációt tartalmazza; L az I/O listát jelenti.

Példa

```
   .
   .
   .
   PUNCH 1Ø, I,J,K
1Ø FORMAT (3I5)
   .
   .
   .
```

A PRINT *utasítás a sornyomtatót jelöli ki kiviteli perifériának.* Hatása azonos a WRITE (*u*,*b*)L utasítás hatásával, ahol *u* a sornyomtatóra írt adatállomány hivatkozási száma. Alakja:

PRINT *b*,L

ahol *b* vagy egy FORMAT utasítás címkéje, vagy egy tömbazonosító, amely a FORMAT specifikációt tartalmazza; L az I/O listát jelöli.

Példa

```
   .
   .
   .
   PRINT 2Ø, X,Y,Z
2Ø FORMAT ('▽X = ',F7.3,'Y = ', F7.3,'Z = ',F7.3)
   .
   .
   .
```

3. Bináris formájú átvitelek. Mágnesszalagkezelő utasítások: BACKSPACE, REWIND, ENDFILE

Az IBM FORTRAN-ban a bináris formájú átvitelek (FORMAT utasítás nélküli átvitelek) csak annyiban különböznek a könyv Hatodik fejezetének 3. alfejezetében leírtaktól, hogy a READ utasítás ERR $= c$ és END $= d$ paraméterei ebben az esetben is használhatók.

A BACKSPACE, REWIND és ENDFILE utasítások is azonosak az ASA FORTRAN megfelelő utasításaival. Meg kell azonban jegyeznünk, hogy ezek az utasítások nemcsak mágnesszalagos, hanem mágneslemezes adatállományok esetén is használhatók. A REWIND utasítás után kiadott első READ vagy WRITE utasítás a mágneslemezes adatállomány első rekordját fogja beolvasni, ill. kiírni. Az ENDFILE utasítás egy end-of-file rekord írását eredményezi, vagyis felír egy jelzést, amely az adatállomány végét jelenti. A BACKSPACE utasítás az adatállományban egy rekorddal való visszalépést jelent.

4. Karakterátvitelek. A FORMAT utasítás

Az IBM FORTRAN a következő formátum specifikációkat ismeri:

Specifikáció	*Jelentése*
I*w*	egész számok konverziója
F*w*.*d*	fixpontos, valós számkonverzió
E*w*.*d*	lebegőpontos, valós számkonverzió
G*w*.*d*	*egész*, valós vagy *logikai* típusú konverzió
D*w*.*d*	kétszeres pontosságú valós számkonverzió
L*w*	logikai mennyiségek átvitele
A*w*	szöveg beolvasása és kiírása
*w*H	szöveg beolvasása és kiírása
'literal'	*szöveg beolvasása és kiírása*
*w*X	helykitöltés
T*r*	*tabulátorbeállítás*
Z*w*	*hexadecimális mennyiségek be- és kivitele*

a) Egész számok konverziója. Az I specifikáció

Ha a kiírandó egész szám számjegyeinek száma nagyobb, mint a megadott mező szélessége, a számjegyek helyett „*" karakterek jelennek meg.

Beolvasáskor a mezőben levő szóköz (space) karakter Ø-vá alakul. (Jobbra tömörítve adjuk meg a számjegyeket, és a számjegyek között ne hagyjunk szóközt! Ez minden specifikációra érvényes!)

b) Fixpontos, valós számkonverzió. Az F specifikáció

Ha a kiírandó mennyiség számára nem elegendő a megadott mező szélessége, a szám helyett „*" karakterek jelennek meg.

c) Lebegőpontos, valós számkonverzió. Az E specifikáció

Input esetén az F és E specifikáció felcserélhető, tehát az E specifikációval beolvasandó adatokat megadhatjuk fixpontos alakban is.

Az E exponenssel megadott adatoknál különösen ügyeljünk arra, hogy az adat jobbra tömörített legyen, ugyanis, ha az exponens rész után például még egy szóközkarakter áll, úgy az exponens 1Ø-zel szorzódik. Például legyen a FORMAT utasításban megadott mező szélessége 1Ø:

```
   |   | .
   |   | .
   |   | .
   |   | READ (6,1Ø) A,B,C
 1Ø|   | FORMAT (4E1Ø.3)
   |   | .
   |   | .
   |   | .
```

A bevitt adatok:

```
|▽▽▽▽▽▽3.14|▽Ø.415E+11|Ø.333E+Ø4▽|
```

Az I/O listán felsorolt változók értéke a következő lesz:

A = Ø.314E+Ø1

B = Ø.415E+11

C = Ø.333E+4Ø

Tehát a C változó értéke hibás, mivel a harmadik adat nincs jobbra tömörítve!

Kivitel esetén, ha a mezőszélességet hibásan úgy adjuk meg, hogy $w<d+5$, a szám helyett „*" karakterek jelennek meg. Ha $w = d+5$ szélességű, a szám csak tizedespontot, d számú tizedesjegyet és decimális exponenst tartalmaz.

Példa

```
      .
      .
      .
      WRITE (6,1Ø) A,B,C
1Ø    FORMAT (1H▽, 'A = ',E6.3, 'B =',E7.3, 'C = ',E8.3)
      .
      .
      .
```

A kiírás eredménye:

```
A = ******B = *******C =.391E Ø1
```

d) Valós számkonverzió előírt számú decimális jegyre. A G specifikáció

Az IBM FORTRAN-ban a G specifikáció az I/O listán szereplő *változók típusának megfelelően*, *egész*, *valós és logikai típusú* mennyiségek be- és kivitelére szolgál.

Kiíratásnál az INTEGER és LOGICAL típusú mennyiségek esetén a specifikáció G*w* alakú lehet (ha G*w*.*d* alakban írjuk, *d*-nek nincs hatása). A nyomtatás az I és L specifikációnak megfelelően történik. REAL típusú mennyiségek írásánál *d* a decimális jegyek számát határozza meg. Ha a kiírandó számot *n*-nel jelöljük és *n* a $Ø.1 \leqq n < 1Ø^d$ tartományba esik, akkor *n* decimális exponens nélkül jelenik meg, különben E vagy D specifikáció szerint íródik ki, a változó hosszától függően.

Beolvasáskor a gép ellenőrzi, hogy az I/O listán szereplő változó és az adat típusa megegyezik-e. Ha a típusok nem egyeznek, a program hibajelzéssel leáll. INTEGER típusú változó esetén a specifikáció G*w* alakú is lehet (ha G*w*.*d* alakban írjuk, *d*-nek nincs hatása). Valós típusú változók esetén a G specifikáció hatása azonos az F, E és D specifikációk hatásával.

Általában a FORTRAN be- és kiviteli rendszere nem ellenőrzi, hogy a beolvasandó vagy kiírandó változó típusa összhangban van-e a formátum specifikációval. Például egy egész típusú változóba F specifikációval beolvasott adat valós számként ábrázolódik (vagyis nem alakul egész számmá és hibajelzést sem kapunk). Az így tárolt értékkel aritmetikai

művelet során mint egész mennyiséggel számol a program, ami természetesen hibás eredményre vezet.

A G specifikáció nagy előnye, hogy beolvasásnál típusellenőrzés történik, kiírásnál pedig a típusnak megfelelő formában jelenik meg a kivitt adat.
Például:

```
   |  .
   |  .
   |  .
   |  DOUBLE PRECISION PI
   |  .
   |  .
   |  .
   |  READ (5,1Ø) B,PI,L,J
 1Ø|  FORMAT (4G1Ø.3)
   |  .
   |  .
   |  .
   |  WRITE (6,2Ø) B,PI,L,J
 2Ø|  FORMAT (1H▽, 4G13.6)
```

A bemenő adatok:

```
|      2.71| 3.14159265|         12|          6|
```

A sornyomtató megjelenő eredmények:

```
|▽ 2.71ØØØ▽▽▽▽▽|▽ 3.14159▽▽▽▽▽|▽▽▽▽▽▽▽▽▽▽▽ 12|▽▽▽▽▽▽▽▽▽▽▽▽6|
```

e) Kétszeres pontosságú mennyiségek átvitele. A D specifikáció

Kétszeres pontosságú mennyiséget E specifikációval is kiírhatunk. A decimális exponens jele ebben az esetben is D.

f) Logikai mennyiségek átvitele. Az L specifikáció

A specifikáció nem tér el a szabványtól.

g) Szöveg beolvasása és kiírása. Az A specifikáció

A specifikáció nem tér el a szabványtól.

h) Szöveg beolvasása és kiírása. A H specifikáció

Hollerith-konstans beolvasása, ill. kiírása kétféle módon történhet: a *w*H specifikációval és *idézőjelek közé zárva*. Például a következő két FORMAT utasítás ekvivalens.

```
25| |FORMAT (21H▽A▽PROGRAM▽EREDMENYEI)
25| |FORMAT ('▽A▽PROGRAM▽EREDMENYEI')
```

Az idézőjelek közé zárt szövegkonstans kiírásának, ill. beolvasásának szabályai megegyeznek a H specifikáció szabályaival. Kivitel esetén az idézőjelek közé zárt szövegkonstans használata kényelmesebb, mert nem kell megszámolni a karakterek számát.

Szövegkonstans olvasásánál a beolvasott karakterek a FORMAT utasításban levő szövegkonstans helyére kerülnek. Ha a FORMAT utasításban a szövegkonstanst idézőjelek közé írtuk, akkor annyi karakter kerül beolvasásra, ahány karakter volt az idézőjelek között.

Például:

```
   | |.
   | |.
   | |.
 1Ø| |FORMAT ('▽▽FEJLEC')
   | |READ (5,1Ø)
   | |.
   | |.
   | |.
```

A fenti programrészlet hatására az input rekord első 8 karaktere a 1Ø-es címkéjű FORMAT utasítás ▽▽FEJLEC karaktereit felülírja.

i) Helykitöltés. Az X specifikáció

A specifikáció nem tér el a szabványtól.

j) Új rekord megnyitása. A vessző és a ferde törtvonal mint elhatároló jelek

A specifikáció nem tér el a szabványtól.

k) Vezérlő karakterek

A sornyomtatóra küldött rekord első karaktere, a sornyomtatót vezérlő karakter, nem jelenik meg a kimeneten! Ennek a szabálynak a megszegése hibajelzést nem okoz, de a nyomtatási kép megváltozhat. A sor első karaktere minden esetben elvész, és attól függően, hogy mi volt az első karakter, több soremelés vagy lapváltás következhet be. A vezérlő karakterek a következők:

Karakter	*Jelentése*
▽ (szóköz)	nyomtatás előtt 1 sormelés
Ø	nyomtatás előtt 2 soremelés
1	lapváltás
+	nyomtatás előtt nincs soremelés

Ez a szabály csak sornyomtatóra vonatkozik. Egyéb periféria esetén a rekord első karaktere adatként értelmeződik.

l) Decimális normálótényező. A P kiegészítő specifikáció

A specifikáció nem tér el a szabványtól.

m)* Tabulátorbeállítás

Ez a specifikáció a szerkesztési specifikációk közé tartozik, a rekordnak azt a pozícióját határozza meg, ahonnan vagy ahová az átvitel történik. Alakja

T*r*

ahol *r* a rekordpozíciót jelöli. Sornyomtatón való nyomtatásnál azonban az első karakter a sornyomtatót vezérli, ezért *r* értékét mindig eggyel nagyobbra kell választani, mint a kívánt karakterpozíció.

Egy FORMAT utasításban több T specifikációt lehet megadni. A T specifikációk sorrendje tetszőleges lehet, mivel *r* mindig az első pozíciótól számított pozíciót határozza meg.

Példa sornyomtatón való írásra

```
      WRITE (6,1Ø)
 1Ø   FORMAT (T4Ø,'1983 STATISZTIKAI JELENTES',
     *T8Ø,'DECEMBER',T2,'SORSZAM 1ØØ')
```

A nyomtatási kép:

```
1. pozíció      39. pozíció                  79. pozíció
↓               ↓                            ↓
SORSZAM 1ØØ     1983 STATISZTIKAI JELENTES   DECEMBER
```

Példa olvasásra

Legyenek adataink lyukkártyán, a következő elrendezésben:

14.62		13.75	AR	215.95
←5→	←4→	←5→	←6→	←6→

Ezekből az adatokból csak a harmadik és ötödik adatra van szükségünk. A beolvasás történhet a következő módon:

```
      READ (5,2Ø) A,B
 2Ø   FORMAT (T1Ø,F5.2,T21,F6.2)
```

Az A és B változók értéke rendre 13.75 és 215.95 lesz.

n)* Hexadecimális mennyiségek átvitele. A Z specifikáció

Hexadecimális mennyiségek átvitelére az IBM FORTRAN-ban a Z*w* specifikációt használhatjuk.

Kivitel esetén, ha a változóban tárolt hexadecimális mennyiség számjegyeinek száma kisebb *w*-nél, a számjegyek a kimeneti mezőben jobbról tömörítve jelennek meg. A fel nem használt karakterpozíciók szóközkarakterekkel töltődnek fel. Ha a kiírandó mennyiség számjegyeinek száma nagyobb *w*-nél, a legnagyobb helyiértékű jegyek nem íródnak ki.

Beolvasáskor a mező bármelyik pozícióján álló szóközkarakterek Ø-t jelentenek. A tár egy bájtjában 2 hexadecimális számjegyet lehet tárolni. Ezért, ha a beolvasandó mezőben páratlan számú számjegy áll, beolvasáskor a szám balról egy

Ø-val egészül ki. Ha a beolvasandó mezőben megadott szám jegyeinek száma nagyobb, mint amennyi a változóban elfér, a legmagasabb helyiértékű jegyek elvesznek.

Példa

```
      INTEGER*2 K
      .
      .
      .
      READ (5,1Ø) A,B,K
1Ø    FORMAT (Z8,F5.2,Z4)
      .
      .
      .
5     WRITE (6,2Ø) A,K
2Ø    FORMAT (1H▽,'A=',Z6,2X,'K=',Z6)
```

Legyen a beolvasandó rekord a következő:

			7	F	1	6	▽	—	3	.	1	2	F	F	1	1

Beolvasás után A = ØØØ7F16Ø, B = —3.12, K = FF11 lesz. Az 5-ös címkéjű WRITE utasítás (ha közben a változók értékét nem változtattuk meg) hatására a következő sor jelenik meg a sornyomtatón:

A = ▽7F16Ø▽▽K = ▽▽FF11

5. A FORMAT és az I/O lista kölcsönhatása. FORMAT lista elhelyezése tömbökben

FORMAT utasítást használó READ és WRITE IBM FORTRAN utasítások végrehajtásakor a FORMAT lista értelmezése a szabvány szerint történik.

A FORTRAN nyelvben a READ és WRITE karakterátviteli utasításokban egy FORMAT utasítás címkéje helyett tömbazonosítót is használhatunk. Ez „változó" formátum specifiká-

cióra ad lehetőséget. A FORMAT listát beolvashatjuk egy tömbbe, vagy DATA *utasításban és explicit típusdeklaráló utasításban* is megadhatjuk. A következő szabályokat kell figyelembe venni:

— a FORMAT listát mindig tömbben kell elhelyezni, még akkor is, ha a tömb dimenziója 1;

— a FORMAT listának ugyanolyan alakúnak kell lennie, mint a forrásprogram FORMAT utasításának, kivéve, hogy a FORMAT szót és az utasításcímkét nem kell kiírni;

— a FORMAT lista tartalmazhat H *specifikációt vagy idézőjelek közé zárt szövegkonstanst;*

— az olyan FORMAT lista, amelyben az idézőjelek közé zárt szövegkonstansban az idézőjel karakter is előfordul, csak írásra használható;

— ha beolvasásra adunk meg olyan FORMAT listát, amelyben a szövegkonstans idézőjelet is tartalmaz, akkor a H specifikációt kell használni.

Példa

```
      DIMENSION FMT (5), C(5)
 1    FORMAT (5A4)
      READ (5,1) FMT
 C    A FORMAT LISTA OLVASASA
      READ (5,FMT) A,B, (C(I), I = 1,5)
 C    OLVASAS TOMBBEN ELHELYEZETT FORMATUMMAL
      .
      .
      .
```

A beolvasott adat első rekordja:

(2E1Ø.3, 5F1Ø.6)

A második rekordnak az A, B változók és a C tömb elemeinek értékét kell tartalmaznia.

Megadhatjuk a FORMAT listát kezdeti értéket adó utasításban is.

```
DIMENSION FMT (1Ø), C(5)
REAL FMT/33H('▽AZ EREDMENYEK:',2E1Ø.3,5F1Ø.6)/
.
.
.
WRITE (6,FMT) A,B, (C(I), I = 1,5)
.
.
.
```

a)* Karakterátvitel a NAMELIST utasítás használatával

Az IBM FORTRAN lehetőséget nyújt kötetlen formájú adatok beolvasására, ill. kiírására. A NAMELIST utasítással változók és tömbnevek egy csoportjához szimbolikus nevet rendelünk és a READ, ill. a WRITE utasításokban ezekre a változócsoportokra hivatkozunk.

A NAMELIST utasítás alakja:

NAMELIST/X_1/$lista_1$/X_2/$lista_2$/. . ./X_n/$lista_n$

ahol X_i egy szimbolikus név, az ún. NAMELIST név. X_i nem lehet azonos a programszegmensben használt változó- vagy tömbnevek egyikével sem. A NAMELIST nevet két törtvonal közé kell zárni; $lista_i$ az ún. NAMELIST lista, változónevekből vagy tömbnevekből áll. Az azonosítókat vesszővel kell egymástól elválasztani. A változók vagy tömbnevek listáját vagy egy újabb NAMELIST név, vagy a NAMELIST utasítás vége zárja le.

Egy változónév vagy tömbnév több NAMELIST listában is előfordulhat. A NAMELIST nevet a NAMELIST utasításban kell deklarálni. Egy NAMELIST név csak egyszer fordulhat elő NAMELIST utasításban; ezenkívül csak READ és WRITE utasításokban használható. A NAMELIST utasítás deklarációs utasítás, a programszegmensben az utasításfüggvények előtt, ill. az első végrehajtható utasítás előtt kell állnia. Formális paraméterek a NAMELIST utasítás listájában nem fordulhatnak elő. BACKSPACE utasítás nem adható ki arra az adatállományra, amelyet NAMELIST használatával írtunk vagy olvastunk.

Példa NAMELIST *utasításra*

```
NAMELIST/NEV1/A,B,C/NEV2/I,J,K,L,A
```

A példában két NAMELIST nevet deklaráltunk, a NEV1 és NEV2-t. A NEV1 NAMELIST névhez az A,B,C változókból álló NAMELIST lista, a NEV2-höz pedig az I,J,K,L,A változókból álló lista tartozik.

A NAMELIST utasítást a READ(*a*,*x*) és a WRITE(*b*,*x*) utasításokkal összekapcsolva használjuk, ahol *a* és *b* az adatállományok hivatkozási száma, *x* pedig egy NAMELIST név. Az ilyen, a NAMELIST utasításra hivatkozó READ és WRITE utasításokkal adatokat olvashatunk be, ill. írhatunk ki anélkül, hogy az I/O listát és a FORMAT utasítást megadnánk.

b)* Olvasás NAMELIST használatával

Példa

```
      DIMENSION A(2)
      REAL*8 D,D1
      NAMELIST /X/ C,J,B,D,D1 /Y/ A
  1   READ(5,X)
  2   READ(5,Y)
      .
      .
      .
```

A beolvasandó adatok az 1-es címkéjű READ utasításhoz legyenek pl. a következők:

```
▽ &X C = 5.6234,J = 25,
▽D = 2.45975DØ,B = 6.25E2,D1 = Ø.435D2,&END
```

A 2-es címkéjű READ utasításhoz pedig:

```
▽ &Y A = 'ABCDEFGH',&END
```

Az adatokat a második karakterpozíción kezdtük.

A példából látható, hogy egy NAMELIST név meghatároz egy adatcsoportot, melyet a NAMELIST utasításban deklarált névvel, &X-el kezdtünk és &END-del zártunk le. Az adatcsoporton belül az adatokat tetszőleges sorrendben, egymástól vesszővel elválasztva írtuk.

A beolvasandó adatokat a következő szabályok szerint kell megadni:

— Minden rekord első pozícióján szóközkarakternek kell lennie.

— Az első rekord második pozícióján egy & karakternek kell állnia, amelyet a NAMELIST névnek kell követnie (pl. &NEV1).

— A NAMELIST név után legalább egy szóközkarakternek kell következnie.

— A szóközkarakter(ek) után következnek az adatelemek vesszővel elválasztva. Az utolsó adatelem után nem kötelező vesszőt írni.

— Az adatcsoport végét az &END karaktercsoport zárja le.

Az adatelemek alakja:

név = konstans

A *név* lehet egy változó neve vagy tömbelem. A tömbelem indexe csak pozitív egész szám lehet. A *konstans* lehet egész, valós, komplex, logikai vagy szövegkonstans. A konstansok és a változók vagy tömbelemek típusának meg kell egyeznie. Ez alól csak a REAL*8 típusú (vagy DOUBLE PRECISION) változóhoz rendelt konstans lehet kivétel, amit megadhatunk egyszeres pontosságúként is. A logikai típusú konstans formája lehet .TRUE. vagy T és .FALSE. vagy F.

Az adatelemeket megadhatjuk még a

tömbnév = konstanslista

formában is. A *konstanslistában* egész, valós, komplex, logikai vagy szövegkonstansokat sorolhatunk fel, vesszővel elválasztva. A konstansok száma kisebb vagy egyenlő lehet a tömb elemeinek számával. Egy konstans többszöri előfordulását ismétlési tényezővel jelölhetjük, mint a kezdőérték adó utasításokban.

Az adatelemekben szereplő változóneveknek vagy tömbneveknek elő kell fordulniuk a NAMELIST listában. Az adatelemek tetszőleges sorrendben írhatók. Nem kell minden, a NAMELIST listán megadott változónak, ill. tömbnek értéket adni.

A beolvasandó adatok több rekordból állhatnak. Minden új rekordnak szóközkarakterrel kell kezdődnie, amelyet egy változónév vagy tömbnév követ. Ha az adatok több rekordból állnak, a rekord végén a „, " karaktert elhagyhatjuk. A beolvasandó rekordban a nevek és a konstansok nem tartalmazhatnak szóközkaraktert. A konstansok után álló szóközkarakterek Ø-ként értelmeződnek!

A READ (a, x) utasítás hatására a program beolvassa az adatokat, ellenőrzi, hogy a READ utasításban szereplő NAMELIST név megegyezik-e az olvasott rekordban írt névvel.(Ha nem egyezik meg, a következő adatcsoportot fogja beolvasni). Továbbá ellenőrzi, hogy az adatelemekben szereplő változó- és tömbnevek rajta vannak-e a NAMELIST listán.

Példa

```
COMPLEX C
DIMENSION W(3)
LOGICAL B,L
DOUBLE PRECISION D
NAMELIST/X/ A,B,C/Y/D,I,L/Z/W
.
.
.
READ (5,X)
READ (5,Y)
READ (5,Z)
.
.
.
```

A beolvasandó adatok legyenek a következők:

```
▽ &X
▽A = 11.11,B = .TRUE.,C = (2.Ø, 3.Ø)
▽ &END
▽ &Y
▽D = 3.141592653589793,I = 9,L = .FALSE.
▽ &END
▽ &Z
▽W = —Ø.13482E9,14.875,65.1294
▽ &END
```

c)* Írás NAMELIST használatával

NAMELIST használatával kiírt adatok formátuma olyan, hogy azokat NAMELIST-tel visszaolvashatjuk. A kimenet első rekordjának 2. pozícióján egy „&" karakter, majd a NAMELIST név jelenik meg. Ezután következnek a NAMELIST listában való felsorolási rendjük szerint az adatelemek

név = konstans

alakban. A kiírás végét &END karaktersorozat zárja le. Az adatelemek a típusuknak megfelelő módon íródnak ki. Az egyes adatok mezőszélessége olyan lesz, hogy a változóban tárolt szám összes értékes jegye ábrázolható legyen (fixpontos vagy decimális exponensű formában).

A mennyiség típusa	*Mezőszélesség*	*A kiírt jegyek száma*
kétszeres pontosságú valós	23	16
egyszeres pontosságú valós	15	8
egész	11	1Ø
logikai	1	—

Példa

```
COMPLEX C
DIMENSION W(3)
LOGICAL B,L
DOUBLE PRECISION D
NAMELIST/X/A,B,C/Y/D,I,L/Z/W
.
.
.
WRITE (6,X)
WRITE (6,Y)
WRITE (6,Z)
.
.
.
```

Tegyük fel, hogy a programban a változóknak az előző példában a NAMELIST utasítás használatával beolvasott értékeit nem változtattuk meg. Ekkor a kimenet képe a következő lesz (a helyhiány miatt a mezőszélességnek megfelelő összes szóközkaraktert nem írtuk ki).

```
&X
A = 11.11ØØØØ,B = T, C = (2.ØØØØØØØ,3.ØØØØØØØ)
&END
&Y
D = 3.141592653589793, I =      9, L = F
&END
&Z
W =—Ø.13482ØØØE Ø9, 14.875ØØØ , 65.1294ØØ
&END
```

d)* Közvetlen hozzáférésű adatátvitel és utasításai

Az IBM FORTRAN-ban lehetőség van közvetlen hozzáférésű (direct access) adatállományok feldolgozására. A közvetlen hozzáférésű adatátviteli utasítások lehetővé teszik, hogy a programozó tetszőleges sorrendben férjen hozzá az adatállomány rekordjaihoz írás vagy olvasás céljából (ellentétben a

soros adatátviteli utasításokkal, amelyekkel az adatállomány rekordjai csak elhelyezkedésük sorrendjében, egymás után dolgozhatók fel). Bármelyik rekord feldolgozását követheti bármelyik, tetszőlegesen megválasztott másik rekord feldolgozása.

Közvetlen hozzáférésű adatállományokat csak mágneslemezen lehet létrehozni.

A közvetlen hozzáférésű adatállományban minden rekordhoz tartozik egy azonosító szám, a rekord sorszáma. Az adatállomány létrehozásakor az operációs rendszer számára meg kell mondani annak jellemzőit; hány és milyen hosszúságú rekordot kívánunk elhelyezni az állományban, formátummal vezérelt adatátvitelt fogunk-e használni. Ezen információk birtokában az operációs rendszer létrehozza az adatállományt, kijelöli és sorszámmal látja el a rekordok helyét. Ez a sorszám lesz a rekordazonosító. Minden közvetlen hozzáférésű adatállományhoz egy változó, az ún. hozzárendelt változó tartozik, amelyben a rendszer az átviteli utasítások végrehajtása után a fizikailag következő rekord sorszámát helyezi el.*

A feldolgozást négy utasítás, a READ, WRITE adatátviteli, az adatállomány jellemzőit leíró DEFINE FILE és a rekord megkeresésére szolgáló FIND utasítás támogatja. A READ, WRITE és FIND utasításokban nemcsak az adatállomány hivatkozási számát, hanem annak a rekordnak az azonosítóját (sorszámát) is meg kell adni, amelyet olvasni, írni vagy kiválasztani akarunk.

Ha a READ, WRITE vagy FIND utasítások és az adatállományt leíró DEFINE FILE utasítás különböző programszegmensekben vannak, akkor a hozzárendelt változót vagy COMMON utasításban kell elhelyezni, vagy a SUBROUTINE, ill. FUNCTION szegmens paraméterlistájára fel kell venni. Ha a hozzárendelt változó szubrutin vagy függvény formális paramétere, akkor név szerint kell átvenni (a formális paramétert két „/" karakter közé kell zárni).

* Meg kell jegyezni, hogy a rekordazonosító és a hozzárendelt változó szerepe azonos. A rekordazonosítót a programozó adja meg, a hozzárendelt változó értéket az operációs rendszer állítja be.

e)* A DEFINE FILE utasítás

A közvetlen hozzáférésű adatállomány leírására szolgál. Az utasítás alakja:

DEFINE FILE $a_1(m_1,r_1,f_1,v_1),\ldots,a_n(m_n,r_n,f_n,v_n)$

ahol a_i az adatállomány hivatkozási száma; m_i egész típusú konstans, az adatállomány rekordjainak számát határozza meg; r_i egész típusú konstans, a maximális rekordméret, szó (4 bájt) vagy bájt egységekben (az egység típusát az f paraméter határozza meg); f_i a maximális rekordméret egységének típusát határozza meg, valamint azt, hogy az adatállományra vonatkozó READ vagy WRITE utasításokat formátumvezérléssel vagy anélkül használjuk. A következő értékeket veheti fel: L a maximális rekordméret egysége bájt; az átvitel akár karakter (formátumos) akár bináris (formátum nélküli) lehet; E a maximális rekordméret egysége bájt; az átvitel csak karakterátvitel lehet; U a maximális rekordméret egysége szó; csak formátum nélküli átvitel lehet; v_i egész típusú, index nélküli változó, az ún. hozzárendelt változó. Minden írás vagy olvasás után v_i értéke egy mutató lesz, amely az éppen soron következő rekordra mutat. FIND utasítás után v_i a keresett rekordra mutat.

A hozzárendelt változó nem fordulhat elő olyan READ és WRITE utasítások I/O listáján, amelyekben az adatállomány hivatkozási száma megegyezik a DEFINE FILE utasításban megadott hivatkozási számmal.

Példa

```
| | DEFINE FILE 8(5Ø,1ØØ,L,I), 9(1ØØ,5Ø,E,J)
```

Ez az utasítás két adatállományt ír le, az első hivatkozási száma 8, a másodiké 9. Az első adatállomány 5Ø rekordból áll, egy rekordnak a hossza maximálisan 1ØØ bájt lehet. Az adatállományba írhatunk vagy olvashatunk formátummal vagy anélkül, I a hozzárendelt változó. A második adatállomány 1ØØ rekordból áll, a rekordok hossza maximálisan 5Ø bájt, az átvitel csak karakterátvitel lehet, J a hozzárendelt változó.

```
| | DEFINE FILE 1Ø(5Ø,1ØØ,U,K)
```

Az előző utasítás egy 5Ø rekordból álló adatállományt ír le, amelynek hivatkozási száma 1Ø. A rekordok maximális hossza 1ØØ szó (4ØØ bájt), az átvitel csak bináris (formátum nélküli) lehet. K a hozzárendelt változó.

f)* A READ utasítás

Az utasítás alakja READ (*a*'*r*,*b*, ERR = *c*)L

ahol *a* előjel nélküli egész típusú konstans, vagy egész típusú, 4 bájt hosszúságú változó; az adatállomány hivatkozási száma. *a* után egy egyszeres idézőjelnek kell állnia. *r* a rekordazonosító; egész típusú kifejezés, amely meghatározza a rekord relatív helyezetét az adatállományban; *b* nem kötelező paraméter, ha megadjuk, az olvasást vezérlő FORMAT utasítás címkéje, vagy egy tömbazonosító, amely tartalmazza a FORMAT listát; ERR = *c* nem kötelező paraméter, ha megadjuk, *c* egy utasításcímkét jelöl, ahová az olvasás közben bekövetkező hiba esetén a vezérlés átadódik; L az I/O listát jelöli.

Az I/O lista nem tartalmazhatja az adatállományhoz rendelt változót. A közvetlen elérésű adatállományok első rekordjának rekordazonosítója 1. Mielőtt olvasási utasítást adnánk ki egy adatállományra, az adatállományt DEFINE FILE utasításban definiálni kell.

Példa

```
     DEFINE FILE 8(5ØØ,5Ø,L,I1), 9(1ØØ,24,L,I2)
     DIMENSION M(1Ø)
     .
     .
     .
     I2 = 21
     .
     .
     .
 9   FORMAT (5I1Ø)
10   READ (8'16,9) (M(K), K = 1,1Ø)
     .
     .
     .
20   READ (9'I2+5) A,B,C,X,Y,Z
     .
     .
     .
```

A 1Ø-es címkéjű READ utasítás a 8-as hivatkozási számú adatállomány 16. rekordjából kezd olvasni (a rekordazonosító = 16). Az olvasást a 9-es címkéjű FORMAT utasítás vezérli. Az I/O lista szerint 1Ø adatot kell beolvasni, ami két rekordot jelent, mivel a FORMAT utasítás szerint egy rekord 5 adatot (5Ø karaktert) tartalmaz. Az adatállományhoz hozzárendelt változó, I1 értéke az olvasás befejezése után 18 lesz. A 2Ø-as címkéjű READ utasítás a 9-es hivatkozási számú adatállomány 26. rekordjából kezd olvasni (I2+5 = 26). Az átvitel bináris. A 9-es hivatkozási számú adatállományt leíró DEFINE FILE utasításban a maximális rekordméret 24 bájt, az I/O listán szereplő változók helyfoglalása szintén 24 bájt (6×4 = 24), tehát a 2Ø-as címkéjű READ utasítás maximális hosszúságú rekordot olvas. A beolvasás után I2 értéke 27 lesz.

Azt, hogy a READ utasítással hányadik rekordot olvassuk be, a FORMAT utasítás is vezérli. Ha az előző példában a 9-es címkéjű FORMAT utasítás helyett a következőt írjuk:

```
9| |FORMAT (//5I1Ø)
```

akkor a 1Ø-es címkéjű READ utasítás a 16 és 17-es rekordokat átugorja, a 18-as rekordot beolvassa, majd a 19 és 2Ø-as rekordot ismét átugorva, a 21-es rekordot olvassa be. Az adatállományhoz rendelt változó, I1 értéke 22 lesz.

g)* A WRITE utasítás

Az utasítás alakja:

WRITE (*a*'*r*, *b*) L

ahol *a* előjel nélküli egész típusú konstans, vagy egész típusú, 4 bájt hosszúságú változó, az adatállomány hivatkozási száma. *a* után egy egyszeres idézőjelnek kell állnia; *r* a rekordazonosító. Egész típusú kifejezés, amely meghatározza a rekord relatív helyzetét az adatállományban; *b* nem kötelező paraméter; ha megadjuk, az írást vezérlő FORMAT utasítás címkéje, vagy egy tömbazonosító, amely tartalmazza a FORMAT utasítást. L az I/O listát jelöli.

Az I/O lista nem tartalmazhatja az adatállományhoz hozzárendelt változót. Mielőtt egy adatállományba írunk, az adatállományt a DEFINE FILE utasításban definiálni kell. Az új közvetlen hozzáférésű adatállományokat az operációs rendszer az első, az adatállományra kiadott WRITE utasítás végrehajtásakor automatikusan hozza létre.

Példa

```
      DEFINE FILE 1Ø(1ØØ,8Ø,L,I1), 12(1ØØ,4Ø,L,I2)
      DIMENSION A(9) , B(1Ø)
      .
      .
      .
      I2 = 2Ø
      .
      .
      .
1Ø    FORMAT (9F8.3,I8)
2Ø    WRITE (1Ø'1Ø,1Ø) A,K
      .
      .
      .
3Ø    WRITE (12'I2+5) B
```

A 2Ø-as címkéjű WRITE utasítás a 1Ø-es hivatkozási számú adatállomány 1Ø. rekordjába, a 1Ø-es FORMAT utasításnak megfelelően ír egy rekordot. A rekord hossza 8Ø bájt lesz. Az adatállományhoz hozzárendelt változó, I1 értéke az írás után 11 lesz. A 3Ø-as címkéjű WRITE utasítás a 12-es hivatkozási számú adatállomány 25. rekordjába formátum nélkül kiírja a B tömb tartalmát. Az I/O lista szerint a rekordméret 4Ø bájt.

Mint ahogyan azt a READ utasítás leírásánál is láttuk, azt, hogy hányadik rekordba írunk, a FORMAT utasítás is befolyásolhatja.

h)* A FIND utasítás

Az utasítás alakja

FIND (*a'r*)

ahol *a* az adatállomány hivatkozási száma; *r* a rekordazonosító.

A FIND utasítás paramétereire ugyanazok a szabályok érvényesek, mint a READ és WRITE utasítás megfelelő paramétereire.

Egy előző READ utasítással beolvasott rekord feldolgozása alatt a FIND utasítás megkeresi az *r* azonosítójú rekordot. Az így megkeresett rekord csak egy következő READ utasítás hatására lesz hozzáférhető. A FIND utasítás az adatállományhoz hozzárendelt változó értékét *r* értékével tölti fel. Ezzel az utasítással a program végrehajtási sebességét növelhetjük. Írás esetén használata semmi előnyt nem jelent.

Példa

```
      DEFINE FILE 13(1ØØØ,1ØØ,L,I)
1Ø    FIND(13'5Ø)
      .
      .
      .
15    READ(13'5Ø) A,B
      .
      .
      .
```

A GYAKORLATOK MEGOLDÁSAI

Második fejezet

1. *a)* `SQRT((1.0-COS(AL+BL))/(1.0+COS(AL+BL))*`
`(1.0-COS(AL-BL))/(1.0+COS(AL-BL)))`

b) `((1.0+AL)**N-1.0)/(1.0+AL)**N/AL`

c) `EXP(-(X-A)**2/2.0/S**2)/SQRT(2.0*PI*S**2)`

d) `SQRT((AK-1.0)*4.0*AL**2*G*H/AK/(1.0+`
`4.0*H**2*(AL**2+BL**2))`

e) `A*EXP(Z/2.0/H)*SIN(AL*X)*COS(BL*Z)*COS(O*T)`

2. *a)* `A/B/(-C)+A/B*(C**2)**3`, vagy
`A/B/(-C)+A/B*C**(2**3)`

b) `SIN(X**2)+(SIN(X))**2`

c) `A*E-2.0**B` vagy
`A*1E-2**B` vagy
`(A*1E-2)**B` vagy
`AE-2.0**B`

d) `(-B+SQRT(B**2-4.0*A*C))/2.0/A`

e) `C**(A/2.0)-FLOAT(I)` vagy
`C**A/2.0-FLOAT(I)`

3. *a)* $\frac{a \cdot a_1}{b \cdot b_1} \sqrt{1+\cos(2x)}$

b) $\sqrt[3]{-q+\sqrt{q^2+p^3}}$

4. `I-(I/2)*2`

5. *a)* `((I+1)/2)*2`

b) `1+(I/2)*2`

c) `1+((I-1)/2)*2`

6. I−(I/M)*M

7. *a)* I1+I2−((I1+I2−1)/M)*M

b) I1−I2+((M−I2+I1)/M)*M

8. IABS(I−(I/2)*2)

I+IABS(I−(I/2)*2)

I+IABS(I+1−((I+1)/2)*2)

I−IABS(I+1−((I+1)/2)*2)

9. AINT((B−A1)/D)*D+A1

10. *a)* AMIN1(A,B−A)

b) AINT(A/B)+AINT((B−A)/B)

11. AMIN1(ABS(AMOD((B+PI/2.0), (2.0*PI))), (2.0*PI
−ABS(AMOD((B+PI/2.0), (2.0*PI))))−PI/2.0)

12. *a)* FALSE

b) TRUE

c) TRUE

d) FALSE

e) TRUE

13. FI.LE.PI/2.0.AND.FI.GE.0.0.OR.FI.GE.(−2.0*PI).AND.
FI.LE.(−1.5*PI)

14. 2**INT(ALOG(A)/ALOG(2.0))

Harmadik fejezet

1.

```
      P=0.0
      DO 1 I=1,N
    1 P = P+X(I)
      P=P/FLOAT(N)
      S=0.0
      DO 2 I=1,N
    2 S = S+(X(I)−P)**2
      S = S/FLOAT(N−1)
```

2.

```
      S1=0.0
      S2=0.0
      S3=0.0
      S4=0.0
      S5=0.0
      DO 1 I=1,N
      T=X(I)
      U=Y(I)
      S1 = S1+T
      S2 = S2+T**2
      S3 = S3+U
      S4 = S4+T*U
    1 S5 = S5+U**2
      XN=FLOAT(N)
      D = XN*S2-S1**2
      B = (S2*S3-S1*S4)/D
      A = (XN*S4-S1*S3)/D
      Q = S5-2.0*A*S4+A**2*S2-2.0*B*S3+
     12.0*A*B*S1+XN*B**2
```

3.

```
      I=1
      O1=0.1
      X=O1
      DO 10 JX=1,10
      XX = X**2
      Y=O1
      DO 9 JY=1,10
      YY = Y**2+XX
      XY = X*Y
      Z=O1
```

```
      DO 8 JZ=1,10
      ZZ = Z* *2+YY
      XYZ = XY*Z
      F(I)=ZZ/XYZ
      I = I+1
 8    Z = Z+O1
 9    Y = Y+O1
10    X = X+O1
```

Amennyiben a függvényértékeket egy háromindexes tömbbe helyezzük, a függvény szimmetriája is kihasználható. Az olvasóra bízzuk egy ilyen programváltozat kidolgozását.

4. Legyenek az (x_1^0, x_2^0,... x_N^0) értékek az X N-elemű vektorban elhelyezve. Ekkor egy lehetséges megoldás:

```
      AT=0.0
      BT=0.0
      DO 1 J=1,N
      AT = AT+A(J)*X(J)
 1    BT = BT+B(J)*X(J)
      IF(BT)2,3,2
 2    F=AT/BT
      GO TO 4
 3    F = -1000.0
 4    „kiírás"
      DO 20 J=1,N
      A1 = A(J)*X(J)
      B1 = B(J)*X(J)
      A2 = AT-A1
      B2 = BT-B2
      K=I(J)
      DO 19 KK=1,K
```

```
      AT = A2+A(J)*DAT(KK,J)
      BT = B2+B(J)*DAT(KK,J)
      IF(BT)12,13,12
   12 F=AT/BT
      GO TO 14
   13 F = -1000.0
   14 „kiírás"
   19 CONTINUE
      AT = A1+A2
   20 BT = B1+B2
```

5.

```
      N1 = N-1
      N2 = N+1
      DO 10 I=1,N1
      I1 = I+1
      W=1.0/A(I,I)
      DO 9 K=I1,N2
    9 A(I,K) = W*A(I,K)
      DO 10 J=I1,N
      W=A(J,I)
      DO 10 K=I1,N2
   10 A(J,K) = A(J,K)-W*A(I,K)
      X(N)=A(N,N2)/A(N,N)
      DO 11 I=1,N1
      J = N-I
      X(J)=0.0
      J1 = J+1
      DO 12 K=J1,N
   12 X(J) = X(J)+A(J,K)*X(K)
   11 X(J) = A(J,N2)-X(J)
```

6.

```
      DO 50 J=1,JMAX
      Z=0.0
      KJ=0
      DO 20 I=1,N
      X1(I)=B1(I)
      DO 19 K=1,N
   19 X1(I) = X1(I)+P(I,K)*X(K)
      H = ABS(X1(I)-X(I))
      IF(H-EPS)20,2,2
    2 KJ = KJ+1
   20 Z = Z+H**2
      DO 15 I=1,N
   15 X(I)=X1(I)
      IF(KJ)16,16,50
   16 IF(Z-EPS1)75,50,50
   50 CONTINUE
   51 „folytatás"
```

A programrészlet két konvergenciafeltétel egyidejű teljesülését vizsgálja:

1. $|x_I^{(k+1)} - x_I^{(k)}| \leqq$ EPS teljesüljön minden I-re.

2. $\sum_{i=1}^{N} (x_I^{(k+1)} - x_I^{(k)})^2 <$ EPS1, a „hibavektor" abszolút értékének négyzete is legyen elegendően kicsi. Kellő konvergencia esetén a 75-ös címkére, ellenkező esetben az 51-es címkére adódik a vezérlés.

7.

```
      DO 50 J=1,JMAX
      Z=0.0
      KJ=0
      DO 20 I=1,N
      OB=B(I)
      OV=X(I)
      X(I)=0.0
```

```
      DO 19 K=1,N
   19 OB = OB-A(I,K)*X(K)
      OB=OB/A(I,I)
      H = ABS(OB-OV)
      IF(H-EPS)28,2,2
    2 KJ = KJ+1
   28 X1(I)=OB
      X(I) = OV
   20 Z = Z+H**2
      DO 15 I=1,N
   15 X(I)=X1(I)
      IF(KJ)16,16,50
   16 IF(Z-EPS1)75,50,50
   50 CONTINUE
   51 „folytatás"
```

A programrészlet a 6. feladat megoldása szerint készült

8.

```
      DO 50 J=1,JMAX
      Z=0.0
      KJ=0
      DO 20 I=1,N
      OB=B(I)
      DO 19 K=1,N
      IF(I-K)3,19,3
    3 OB = OB-A(I,K)*X(K)
   19 CONTINUE
      OB=OB/A(I,I)
      H = ABS(OB-X(I))
      IF(H-EPS)28,2,2
    2 KJ = KJ+1
```

```
28    X(I)=OB
20    Z = Z+H**2
      IF(KJ)16,16,50
16    IF(Z-EPS1)75,50,50
50    CONTINUE
51    „folytatás”
```

A programrészlet a 6. és 7. megoldások szerint készült.

9. A kért programrészlet az előző gyakorlatok szellemében:

```
      DO 50 J=1,JMAX
      Z=0.0
      KJ=0
      DO 20 I=1,N
      OB=B(I)
      DO 19 K=1,N
      IF(I-K)3,19,3
3     OB = OB-A(I,K)*X(K)
19    CONTINUE
      OB=OB/A(I,I)
      OB = OMEGA*OB+(1.0-OMEGA)*X(I)
      H = ABS(OB-X(I))
      IF(H-EPS)28,2,2
2     KJ = KJ+1
28    X(I)=OB
20    Z = Z+H**2
      IF(KJ)16,16,50
16    IF(Z-EPS1)75,50,50
50    CONTINUE
51    „folytatás”
```

10. Tegyük fel, hogy a kezdőértékek az N-elemű X tömbben és az ALAM változóban vannak elhelyezve. A programrészlet:

```
      DO 50 I=1,ITMAX
      YMAX=0.0
      DO 9 J=1,N
      Y(J)=0.0
      DO 3 K=1,N
    3 Y(J) = Y(J)+A(J,K)*X(K)
      IF (YMAX.LT.ABS(Y(J))) MAX = J
    9 YMAX=ABS(AMAX1(YMAX,ABS(Y(J))))
      DO 4 J=1,N
      PL=Y(J)/X(J)
      Z = ABS(PL/ALAM-1.0)
      IF(Z-EPS)4,4,7
    4 CONTINUE
      ALAM=Y(MAX)/X(MAX)
      S=0.0
      DO 5 J=1,N
    5 S = S+Y(J)**2
      S=SQRT(S)
      DO 6 J=1,N
    6 Y(J)=Y(J)/S
      GO TO 25
    7 ALAM = Y(MAX)/X(MAX)
      DO 50 J=1,N
   50 X(J)=Y(J)/ALAM
   51 ...
```

Ez a programrészlet sikeres iteráció esetén a 25-ös címkére, ha pedig a konvergencia nem megfelelő, az 51-es címkére adja a vezérlést. (Némileg módosul ez a programrészlet, ha memóriatakarékosság céljából az A mátrix főátló alatti elemeit más mennyiségek tárolására használjuk.)

11.

```
      DO 1 J=1,N
      DO 1 K=1,N
    1 V(J,K)=0.0
      DO 2 J=1,N
    2 V(J,J)=1.0
      DO 7 I=1,N
      DO 4 J=1,N
      DO 4 K=1,N
      Z=0.0
      DO 3 L=1,N
    3 Z = Z+A(J,L) * V(L,K)
    4 U(J,K)=Z
      Z=0.0
      DO 5 L=1,N
    5 Z = Z+U(L,L)
      Z = -Z/FLOAT(I)
      ALFA(I)=Z
      DO 6 J=1,N
      DO 6 K=1,N
    6 V(J,K)= U(J,K)
      DO 7 J=1,N
    7 V(J,J) = V(J,J)+Z
      Z=0.0
      DO 8 J=1,N
      DO 8 K=1,N
    8 Z = Z+V(J,K)* *2
      IF(Z-EPS)9,9,10
      „folytatás"
```

Ha a számítási hibák elegendően kicsinyek, a 9-es címkére, ellenkező esetben pedig a 10-es címkére adódik a vezérlés.

Negyedik fejezet

1. Példaképpen itt csak a Harmadik fejezet **9.** gyakorlatát látjuk el a szükséges bevezető és végső utasításokkal:

```
   MASTER GAUSS
   DIMENSION X(100), A(100,100)
   beolvasás: JMAX, N, EPS, EPS1
   IF(N-100) 1,1,80
 1 beolvasás: N×N elem az A tömbbe.
   DO 50 J=1,JMAX
   stb. az előző fejezet megoldásai szerint
   :
16 IF(Z-EPS1) 75,50,50
50 CONTINUE
51 kiírás: „nem konvergál az eljárás"
   PAUSE 21
75 kiírás: „az egyenlet megoldása"; X komponensei.
   STOP 22
80 kiírás: „ilyen sok ismeretlennel nem tudok számolni!"
   STOP 23
   END
```

2. *a)* DIMENSION A(100), B(100), R(101)

EQUIVALENCE (A(1), R(1)), (B(1), R(101))

b) DIMENSION A(100), B(100) R(201)

EQUIVALENCE (A(100), R(1)), (B(1), R(201))

3. *a)* DA-hoz és DB-hez csak kétszeres pontosságú számkonstanst lehet hozzárendelni, ezért a DATA utasítás helyesen:

```
  DATA A, B, DA, DB/3.1415926,2.71,
1 2.71D+0,3.14159265358979323846D-0/
```

b) A DATA utasítás több okból hibás:
1. Itt nem lehet az A tömböt egy indexszel írni.
2. De még ha lehetne, akkor sem lehet egy rekesznek két kezdőértéket adni.
3. Valós változónak nem lehet egész típusú értéket adni.
4. A Hollerith-konstans nem lehet hosszabb, mint amekkorának előre jeleztük. Ezért jobb a 10HCINKEFULKE konstans.

c) A szereplő utasításfüggvény több okból hibás:
1. A DATA utasítás után kellett volna elhelyezni.
2. Azonosítója nem lehet tömbazonosító.
3. Logikai típusú függvény értéke nem lehet számérték.

A szereplő DATA utasítás helyes, de FJ(2) értéke ettől még nem TRUE, hanem a felírt szöveg.

4. A kívánt utasításfüggvények:

a)
```
PRK(A,B) = 2.0*(A+B)

PRT(A,B,FI) = A*B*SIN(FI)
```

b)
```
TETRFS(X) = X**2*SQRT(3.0)

TETRTF(X) = X**3*SQRT(2.0)/12.0

HEXAFS(X) = X**2*6.0

HEXATF(X) = X**3

OKTAFS(X) = X**2*2.0*SQRT(3.0)

OKTATF(X) = X**3*SQRT(2.0)/3.0

DODEFS(X) = X**2*3.0*SQRT(5.0*(5.0+SQRT(20.0)))

DODETF(X)= X**3*(15.0+7.0*SQRT(5.0))/3.0

AIKOFS(X) = X**2*5.0*SQRT(3.0)

AIKOTF(X) = X**3*(3.0+SQRT(5.0))*5.0/12.0
```

Célszerűbb azonban x^2 és x^3 együtthatóit egyetlen számmal megadni!

c) Teljesen felesleges ilyen utasításfüggvények felírása, hiszen a 4. táblázat belső standard függvényei közül kiválasztható a kívánt függvény!!

d)
```
SINO(X) = 2.0*X/(1.0+X**2)

COSO(X) = (1.0-X**2)/(1.0+X**2)

TAGO(X) = 2.0*X/(1.0-X**2)
```

Ötödik fejezet

1. Szubrutinszegmens mátrix transzpolálására:

```
      SUBROUTINE TRANSP(A,N)
      DIMENSION A(N,N)
      LIM1 =N-1
      DO 1 I=1, LIM1
      LIM2 = I+1
      DO 1 K=LIM2, N
      W=A(K,I)
      A(K,I)=A(I,K)
    1 A(I,K)=W
      RETURN
      END
```

A szegmenst aktivizáló vezérprogram vázlata:

```
      MASTER VPROG
      DIMENSION V(49)
      DO 1 I=1, 25
      „folytatás"
    1 V(I)=...
      CALL TRANSP(V, 5)
      „folytatás"
      DO 2 I=1, 36
      „folytatás"
    2 V(I)=...
      CALL TRANSP(V, 6)
      „folytatás"
      DO 3 I=1, 49
      „folytatás"
```

```
      V(I)=...
      CALL TRANSP(V, 7)
      „folytatás”
      STOP
      END
```

2. Szubrutin vektor elemeinek sorba rendezésére:

```
      SUBROUTINE ORDER(N)
      INTEGER U, UMIN
      COMMON/VEK/U(200)
      N1 = N-1
      DO 1 I=1, N1
      K = I+1
      UMIN=U(I)
      DO 2 J=K, N
      IF(U(J).GE.UMIN) GO TO 2
      IMIN=J
      UMIN=U(J)
    2 CONTINUE
      U(IMIN)=U(I)
    1 U(I)=UMIN
      RETURN
      END
```

```
      MASTER UNOV
      INTEGER U(200)
      COMMON/VEK/U
      READ (...)N
      READ (...)(U(I), I=1,N)
      WRITE (...)(U(I), I=1,N)
```

```
      CALL ORDER(N)
      WRITE (...)U((I), I=1,N)
      STOP
      ...
      END
```

A MASTER és a SUBROUTINE szegmens között a közös adatmezőn keresztül történő adatcserét választottuk, mert ez kevesebb gépidőt igényel. Az ORDER szubrutin nem annyira „közérdekű", hogy az U vektort érdemes lenne formális paraméterré tenni. Dinamikus indexhatárokat így természetesen nem alkalmazhatunk. A READ és WRITE utasításokban szereplő (U(I), I=1,N) jelsorozatok ún. „belső ciklusok", amelyek segítségével be tudjuk olvastatni az U vektor első N elemét. (A belső ciklusok tárgyalását lásd a Hatodik fejezetben.)

3. Szubrutin vektorok diadikus szorzatának előállítására:

```
      SUBROUTINE DIAD(D,X,Y,N)
      DIMENSION D(N,N), X(N), Y(N)
      DO 1 I=1,N
      DO 1 J=1,N
    1 D(I,J) = X(I)*Y(J)
      RETURN
      END
```

Az előző példával ellentétben itt a D mátrixot, valamint az X és Y vektorokat formális paraméterként deklaráltuk, mivel ez egy általános, gyakran előforduló mátrixművelet programja. A szubrutin rugalmassága érdekében mindegyikre dinamikus indexhatárokat írtunk elő. Vektorok dinamikus deklarálásának ui. akkor van értelme, ha nem akarjuk előre lerögzíteni azt a maximális elemszámot, amellyel a szubrutin működhet, és itt éppen ez a helyzet; ha X(N) helyett mondjuk X(200)-at írnánk, akkor a rutin 200 eleműnél nagyobb vektorok diadikus összeszorzására nem lenne alkalmas.

4. A mátrixkivonás programja:

```
      SUBROUTINE MATSUB(A, B, C, N)
      DIMENSION A(N,N), B(N,N), C(N,N)
      DO 1 I=1,N
      DO 1 J=1,N
    1 C(I,J) = A(I,J)-B(I,J)
      RETURN
      END
```

5. Mátrix maximális abszolút értékű sajátértékének meghatározása:

```
      SUBROUTINE MAXEIG(A,N,EPS,X,Y,ITMAX,ALAM,SIKER)
      LOGICAL SIKER
      DIMENSION A(N,N), X(N), Y(N)
      SIKER=.TRUE.
      ALAM=1.0
      DO 50 I=1, ITMAX
      YMAX=0.0
      DO 9 J=1,N
      Y(J)=0.0
      DO 3 K=1, N
    3 Y(J) = Y(J)+A(J,K)*X(K)
    9 YMAX=ABS(AMAX1(YMAX, ABS(Y(J))))
      UL=Y(1)/X(1)
      DO 4 J=1, N
      PL=Y(J)/X(J)
      Z = ABS(PL/ALAM-1.0)
      IF(Z-EPS)8, 8, 7
    8 UL=PL
    4 CONTINUE
      ALAM=UL
      S=0.0
      DO 5 J=1,N
    5 S = S+Y(J)**2
      S=SQRT(S)
      DO 6 J=1, N
    6 Y(J)=Y(J)/S
   60 RETURN
    7 DO 50 J=1,N
```

```
50    X(J)=Y(J)/YMAX
51    SIKER=.FALSE.
      RETURN
      END
```

A program — azoktól az eltérésektől eltekintve, amelyeket a szubrutinszegmenssé való kiegészítés megkövetel —, megegyezik a Harmadik fejezet **10.** feladatának megoldásával. Ez a kiegészítés többek között szükségessé tette egy „eredményességi" paraméternek („válaszparaméternek"), a SIKER logikai változónak a bevezetését, amelyik a TRUE vagy a FALSE értéket aszerint veszi fel, hogy az előírt számú iterációs lépésben sikerült-e meghatározni a keresett sajátértéket és a hozzá tartozó sajátvektort vagy sem (vö. pl. (5.72)-vel).

6. Szimmetrikus mátrix összes sajátértékének meghatározása:

```
      SUBROUTINE ALLEIG(AUX1,AUX2,A,N,EPS,X,Y,ITMAX,
     X                ALAM,SIKER)
      LOGICAL SIKER
      DIMENSION AUX1(N,N), AUX2(N,N), A(N,N), Y(N,N),
     X          X(N), ALAM(N)
      SIKER=.TRUE.
      DO 1 I=1,N
      ALAM(I)=1.0
      DO 2 J=1,N
2     X(J)=1.0
      CALL MAXEIG(A,N,EPS,X,Y(1,I), ITMAX, ALAM(I), SIKER)
      IF(.NOT.SIKER) RETURN
      CALL DIAD(AUX1, Y(1,I), Y(1,I), N)
      R0=ALAM(I)
      DO 5 J=1,N
      DO 5 K=1,N
5     AUX1(J,K)=R0*AUX1(J,K)
      CALL MATSUB(A, AUX1, AUX2, N)
      DO 4 J=1,N
```

```
      DO 4 K=1,N
4     A(J,K)=AUX2(J,K)
1     CONTINUE
      RETURN
      END
```

Példánk elsősorban azt kívánja illusztrálni, hogyan lehet több kisebb szubrutin szegmensből egy általánosabbat összeállítani. Attól, hogy numerikus szempontból a kapott eljárás — a többnyire rendkívül lassú konvergencia miatt — nem ajánlható, e helyen eltekinthetünk. (Kis rendszámú és „jó magaviseletű" mátrixokra mindenesetre működik.) Figyelmet érdemel a MAXEIG és a DIAD szubrutinok hívásában az aktuális paraméterként megadott Y(1,I) indexes változó. Így az Y mátrix I-edik oszlopa aktuális paraméterré vált (vö. pl. (5.51)-gyel). DIAD-ban a második és harmadik aktuális paraméter megegyezik; ez megengedett, mert ezeknek a szubrutin nem ad értéket (lásd 143. old., 7. szabály).

7. Iterációs eljárás valós együtthatójú magasabb fokú algebrai egyenlet összes gyökének numerikus meghatározására:

```
      SUBROUTINE HORNER(A,N,L,X,X0,EPS,SIKER,ITMAX,K,B)
      LOGICAL SIKER
      DIMENSION A(N), X(L), X0(L), B(L)
      I=0
      IF(L.NE.N-1) GO TO 5
      SIKER=.TRUE.
      B(1)=A(1)
      DO 1 I=1, L
      M1 = N-I+1
      X(I)=X0(I)
      DO 2 ITER=1, ITMAX
      C=A(1)
      DO 3 J=2,M1
      B(J) = B(J-1)*X(I)+A(J)
      IF(J.NE.M1) C = C*X(I)+B(J)
3     CONTINUE
```

```
      XE=X(I)
      X(I) = X(I)-B(M1)/C
      IF(ABS((X(I)-XE)/XE).LT.EPS) GO TO 4
    2 CONTINUE
      GO TO 5
    4 DO 1 J=3,M1
    1 A(J-1) = B(J-1)
      RETURN
    5 SIKER=.FALSE.
      K=I
      RETURN
      END
```

A szubrutin lényegében három egymásba skatulyázott ciklusból áll, melyek közül a legkülső (I szerinti) az egyes gyökök meghatározását vezérli; a középső (ITER szerinti) egy adott X(I) gyök „beiterálását" végzi el, végül a legbelső (J szerinti) ciklusban történik a polinom és deriváltja helyettesítési értékének kiszámítása. A SIKER paraméter ismét arra ad választ, hogy ITMAX lépésben sikerült-e valamennyi gyököt előállítani.

8. Transzcendens egyenlet megoldása szelőmódszerrel:

```
      SUBROUTINE SECANT(X,X0,X1,F,EPS,N,ITMAX)
      N=1
      F0=F(X0)
      F1=F(X1)
    1 X = X1-(X1-X0)/(F1-F0)*F1
      IF(ABS((X-X1)/X).LT.EPS) RETURN
      X0=X1
      X1=X
      F0=F1
      F1=F(X)
      N = N+1
      IF(N.GT.ITMAX) PAUSE 77
      GO TO 1
      END
```

MASTER szegmens vázlata a szubrutin kipróbálásához:

```
MASTER TESZT
DATA PI,PIFEL/3.141593, 1.570796/
EXTERNAL FCT
READ (...), EPS ITMAX
CALL SECANT(X, PIFEL, PI, FCT, EPS, N, ITMAX)
WRITE (...) N, X
PAUSE 01
...
END
```

```
FUNCTION FCT(X)
FCT = SIN (X)-X/2.0
RETURN
END
```

A példabeli MASTER szegmensben a pontosságot és a megengedett iterációs lépésszámot adatszalagról (kártyáról) olvastathatjuk be. Eredményként a program a szükséges iterációs lépésszámot és a gyök értékét írja ki.

9. Numerikus integrálás a trapézformula alapján:

```
  SUBROUTINE TRAPEZ (A, B, F, N0, EPS, ITG, N)
  REAL ITG, ITGE
  H = (B-A)/FLOAT(N0)
  ITGE = (F(A)+F(B))*0.5
  N = N0-1
  DO 1 I=1, N
1 ITGE = ITGE+F(A+H*FLOAT(I))
  ITGE = H*ITGE
  N=N0
2 A1 = A+0.5*H
  ITG=0.0
```

```
      DO 3 I=1, N
    3 ITG = ITG+F(A1+H*FLOAT(I-1))
      ITG = 0.5*(H*ITG+ITGE)
      N = 2*N
      IF(ABS ((ITG-ITGE)/ITG).LT.EPS) GO TO 4
      ITGE=ITG
      H = 0.5*H
      GO TO 2
    4 RETURN
      END
```

A szubrutin első része az integrál N0-adik közelítését számítja ki; a 2-es címkével ellátott utasításnál kezdődik a felosztás finomításával foglalkozó programrész. A szubrutin nincs védve a „túliterálás" ellen; vagyis nincs beépítve az előző példákból már ismert ITMAX változó. Ezért rossz tulajdonságú függvények esetén előfordulhat, hogy nagyon sokáig számol és erről közben nem ad „vészjelzést".

10. Numerikus integrálás *Romberg* módszerével:

```
      SUBROUTINE ROMBG(A,B,F,NMAX,EPS,ITG,LA,LB)
      REAL ITG
      LOGICAL LA, LB
      DIMENSION T(10)
      LA=.TRUE.
      LB=.TRUE.
      H = B-A
      KSWTCH=2
      T(1) = (F(A)+F(B))*H*0.5
      N=1
   11 DO 1 I=KSWTCH,NMAX
      A1 = A+0.5*H
      T(I)=0.0
      DO 2 K=1, N
```

```
    2 T(I) = T(I)+F(A1+H*FLOAT(K-1))
      T(I) = 0.5*(H*T(I)+T(I-1))
      IF(ABS((T(I)-T(I-1))/T(I)).LT.EPS)
     X GO TO 8
      N=2*N
    1 H=0.5*H
      DO 3 K=KSWTCH, NMAX
      WORK1=4.0
      WORK2=3.0
      LIM = K-1
      DO 4 J=1, LIM
      I = K-J
      T(I) = (WORK1*T(I+1)-T(I))/WORK2
      IF(ABS((T(I+1)-T(I))/T(I)).LT.EPS) GO TO 8
      WORK1 = 4.0*WORK1
    4 WORK2 = WORK1-1.0
    3 CONTINUE
      LA=.FALSE.
      NMAX = NMAX+1
      IF(NMAX.GT.10) GO TO 10
      KSWTCH=NMAX
      GO TO 11
    8 ITG=T(I)
      RETURN
   10 LB=.FALSE.
      RETURN
      END
```

A szubrutin a trapézformula szerinti közelítéseik kiszámításával kezdődik. Ez az 1-es címkéjű utasításnál ér véget. Ezt követi a Romberg-

mátrix további sorainak kiszámítása. A 3-as címkéjű utasítást követő programrész az NMAX változó növelését intézi; erre csak akkor kerül a vezérlés, ha a 2^{NMAX} osztópontszám nem volt elégséges. Ezt a körülményt LA=FALSE jelzi. Ezután visszatérés következik a 11-es címkére, egyetlen újabb trapézintegrál kiszámítása céljából. Majd egy újabb Romberg-sor generálására kerül sor. A KSWTCH változó kezdetben 2-vel egyenlő, NMAX növelése esetén pedig a mindenkori NMAX-szal, és ez biztosítja, hogy ilyenkor már mindig csak egyetlen újabb trapézintegrál, és egyetlen Romberg-sor számítódjék ki.

A programhoz tartozó MASTER és FUNCTION szegmensek szervezése teljesen hasonló a már ismert feladatokéhoz, így ezek felírásától eltekintünk.

Hatodik fejezet

1. Nagyméretű mátrix transzponálása mágnesszalagon:

```
      SUBROUTINE MTRSPS
      DIMENSION A(200,10), B(200)
      DO 1 I=1, 201, 10
      DO 2 J=1, 200
      READ(5) B
      DO 2 K=1, 10
      KIND = I+K-1
2     A(J,K)=B(KIND)
      DO 3 K=1, 10
3     WRITE(6)(A(J,K), J=1, 200)
1     REWIND 5
      ENDFILE 6
      REWIND 6
      STOP
      END
```

2. File-felújítás programja, a rekord sorszámok növekvő elrendezésének feltételezése nélkül.

A feladat megoldásához két mágnesszalag kezelő segéd szubrutint szerkesztettünk:

```
      SUBROUTINE BSP(L,U)
C     VISSZALEPES AZ U MAGNESSZALAGOS EGYSEGEN
C     L SZAMU REKORDDAL
      INTEGER U
      DO 1 I=1,L
    1 BACKSPACE U
      RETURN
      END

      SUBROUTINE FSP(L,U)
C     ELORELEPES AZ U MAGNESSZALAGOS EGYSEGEN
C     L SZAMU REKORDDAL
      DO 1 I=1, L
    1 READ(U)
      RETURN
      END
```

A két segédszubrutin a mágnesszalag előre-hátra mozgatásának szervezéséhez szükséges. Mindkettő kihasználja azt a lehetőséget, hogy az input-output utasításokban a logikai perifériaszámot nem csak számmal, hanem változóval is meg lehet adni. Az FSP (Forspace, előrelépés) szubrutin az üres I/O listájú bináris READ utasítást alkalmazza a szalag előre léptetésére.

Lássuk ezek után a vezérprogramot:

```
      MASTER UPDT1
      INTEGER PNTR1, PNTR2
      DIMENSION IREK(60), REK1(250), REK2(250)
      READ(5)NREK
      DO 1 I=1, NREK
    1 READ(5) IREK(I), REK2
      PNTR1=0
      REWIND 5
      DO 2 J=1, 60
```

```
  READ(6)N01, REK1
  DO 3 I=1, NREK
  IF(N01-IREK(I))3, 4, 3
3 CONTINUE
  WRITE(7) N01, REK1
  GO TO 2
4 PNTR2=I
  LEPSZ = PNTR2-PNTR1-1
  IF(LEPSZ) 5, 6, 7
5 CALL BSP(-LEPSZ)
  GO TO 6
7 CALL FSP(LEPSZ)
6 READ(5) N02, REK2
  IF(N02.NE.IREK(I)) STOP 07
  WRITE(7) N02, REK2
  PNTR1=PNTR2
2 CONTINUE
  STOP
  END
```

A program mindenekelőtt beolvassa a javító file-ról a javítandó rekordok sorszámait, majd visszacsévéli a javító szalagot. Ezt követi a tulajdonképpeni javítás: a program beolvastatja a javítandó file soronkövetkező rekordját, és megvizsgálja, hogy ennek sorszáma megegyezik-e valamelyik javító rekordéval. Ha igen, a javító szalagot a megfelelő javító rekord elé állítja, majd beolvastatja és kiíratja ezt a javító rekordot. Ha a javítandó file soron következő rekordjának sorszáma egyik javító rekord sorszámával sem egyezik meg, a program az eredeti file-ról beolvasott rekordot íratja ki.

Annak érdekében, hogy a javító file-t a lehető legkevesebbet kelljen előre-hátra mozgatni, a programba beépítettünk egy egyszerű nyilvántartási rendszert arról, hogy az utolsó művelet után a szalag a javító file hányadik rekordja után áll. Ezt tartalmazza a PNTR1 programbeli változó. A PNTR2 változó azt mutatja meg, hogy a szalag hányadik rekordját kell kikeresnünk. Az elvégzendő előre- vagy visszalépések száma éppen (PNTR2 − PNTR1 − 1), és attól függően, hogy ez a szám negatív, pozitív, vagy zérus, BSP-t FSP-t vagy egyiket sem hívjuk. Az eljárás addig ismét-

lődik, amíg 60 rekordot ki nem írtunk a 7-es logikai számú mágnesszalagra.

3. Nem; az utóbbi minden szót új rekordba ír ki, míg az előbbi egyetlen 250 szavas rekordot képez az output szalagon.

4. A Pascal-háromszög kinyomtatásának programja:

```
      MASTER PASCAL
      INTEGER A(20)
      WRITE (2, 10)
      A(2)=2
      A(3)=1
      DO 1 I=4, 20
      M1=1
      M2 = A(2)+1
      LIM = I-1
      DO 2 K=3, LIM
      A(K-2) = M1
      M1=M2
2     M2 = A(K-1)+A(K)
      A(LIM-1) = M1
      A(LIM)=M2
      A(I)=1
1     WRITE(2,11)(A(K), K=1,I)
      PAUSE 01
10    FORMAT(1H1, 47X, 25HA PASCAL HAROMSZOG ELEMEI
     X//, 9X, 1H1//9X, 1H1, 4X, 1H1//9X,
     X1H1, 4X, 1H2, 4X, 1H1)
11    FORMAT(1H0, 4X, 20I5)
      END
```

A program a (6.121) rekurziós formulával csak a háromszög negyedik sorától kezdve dolgozik; az első három sort „kívülről tudja", azaz egyszerűen H specifikációval nyomtattatja ki (lásd a 10-es címkéjű FORMAT

utasítást). Megjegyezzük, hogy a nyomtatási kép, amelyet a program ad, nem túl tetszetős, mert az utolsó sorokban már 10^5 nagyságrendű számok vannak, amelyek az I5 specifikáció miatt összefolynak. Ha 160 karakteres sornyomtató áll a rendelkezésünkre, akkor a 11-es címkéjű FORMAT utasításban I6-ot alkalmazva — és a 10-es FORMAT-ot kissé átalakítva — tetszetősebb nyomtatási képet kaphatunk, és 20 sor helyett felmehetünk 25 sorig.

5. A program nem egy, hanem tíz kártyát olvas be; minden kártyának csak az első 8 karakterét veszi figyelembe, a maradék 72-t ignorálja. Az R vektor elemeit ezen tíz kártya első 8—8 karakterével tölti fel.

6. Sajátvektor és sajátérték meghatározásának programja:

```
      MASTER EIGVLS
      REAL A(25), AUX1(25), AUX2(25), LAMBDA(5),
     X         EIGMTX(25), X(5), FOM(4)
      LOGICAL BOOLE
      DATA X(1), X(2), X(3), X(4), X(5)/5*1.0/
      WRITE(2, 20)
      READ(1, 21) N, FOM
      NELEM = N**2
      READ(1,22)(A(I), I=1, NELEM)
      CALL ALLEIG(AUX1, AUX2, A, N, 0.1E-03, X,EIGMTX,
     X         20, LAMBDA, BOOLE)
      IF(.NOT.BOOLE) GO TO 1
      DO 2 I=1, N
      LIM = I+N*(N-1)
    2 WRITE(2,FOM)(A(K), K=I, LIM, N), (EIGMTX(K),
     X         K=I, LIM, N), LAMBDA(I)
      STOP
    1 WRITE(2,24)
      STOP
   20 FORMAT(1H1, 36X,
     X36H SAJATERTEK MEGHATAROZAS PROGRAMJANAK, 1X,
     X11HKIPROBALASA//, 24X, 12HAZ A MATRIX:,
```

```
     X 20X, 32HA SAJATVEKTOROK ES SAJATERTEKEK://)
21     FORMAT(I5, 4A8)
22     FORMAT(5F8.5)
24     FORMAT(/10X,
     X 37HA SAJATERTEK MEGHATAROZAS SIKERTELEN.)
       END
```

A program — mint a feladat kitűzésében szerepel — adatszalagról beolvasott FORMAT-listával működik. A következő feladattal ellentétben sikertelenség esetén nincs felkészítve az addig kiszámított használható részeredmények megmentésére. Természetesen ez különösebb nehézség nélkül beépíthető a programba, csupán az ALLEIG szegmensbe kell további válaszparaméterként beletenni egy változót, amely megadja az addig meghatározott sajátvektorok számát. Ezután a MASTER szegmens már könnyen kiegészíthető olyan utasításokkal, amelyek ezeket kinyomtatják.

7. Program magasabb fokú polinom összes gyökének kiszámítására:

```
       MASTER SLVTRS
       LOGICAL SUCSS
       DIMENSION A(16), X(15), X0(15), B(15)
       READ(1,20), N, EPS
       L = N-1
       READ(1,21)(A(I), I=1, N)
       READ(1,21)(X0(I), I=1, L)
       WRITE(2,22)(A(I), X0(I),I=1, L)
       WRITE(2,23) A(N)
       CALL HORNER(A,N,L,X,X0,EPS,SUCSS,5,K,B)
       IF(.NOT.SUCSS) GO TO 1
       WRITE(2,24)(X(I), I=1,L)
       PAUSE 01
1      WRITE(2,25) K
       IF(K.LT.2) GO TO 2
       WRITE(2,26)(X(I-1), I=2,K)
       WRITE(2,27)(A(I), I=1,K)
```

```
2       PAUSE 07
20      FORMAT(I5, F10.4)
21      FORMAT(8F10.4)
22      FORMAT(1H1,//,32X,
     X  27HN-ED FOKU ALGEBRAI EGYENLET, 1X,
     X  27HMEGOLDASA HORNER MODSZERREL,//, 45X,
     X  29HA(N+1)+A(N)*X+...+A(1)*X**N=0///,
     X  36X, 12HEGYUTTHATOK:, 25X, 13HKEZDOERTEKEK:,//,
     X  1(36X, 1PE12.5, 24X, E12.5))
23      FORMAT(36X, 1PE12.5)
24      FORMAT(//56X, 7HGYOKOK:,//,1(53X,1PE12.5))
25      FORMAT(//10X, 29HSIKERTELEN GYOKMEGHATAROZAS A,
     X  I3, 12H-IK GYOKNEL.)
26      FORMAT(10X, 27HEDDIG MEGHATAROZOTT GYOKOK:, 15X,
     X  15(1PE12.5/52X))
27      FORMAT(/10X, 31HA REDUKALT POLINOM EGYUTTHATOI:,
     X  11X, 15(1PE12.5/52X))
        END
```

A programban a decimális lebegőpontos számok kinyomtatási formátumának módosítására több helyen is alkalmaztuk az 1P kiegészítő specifikációt. Ezt a módszert főként azok kedvelik, akik a decimális számokat szívesebben olvassák pl. 1.23456E 03 alakban, mint 0.123456E 04 alakban. Ezzel mezőszélességben is nyerhetünk egy karaktert. A 26-os és 27-es FORMAT utasításokban alkalmazott ismétlési csoportok funkciójával kapcsolatban utalunk a (6.104) program 23-as és 26-os FORMAT utasításaihoz fűzött megjegyzéseinkre (lásd 277. old.).

8. Adatfeldolgozó program, vállalati bérhelyzet felmérésére:

```
        MASTER BRPROB
        DIMENSION ALSLIM(50), FLSLIM(50), KLCSSZ(50),
     X            ATLAG(50), NMAX(50), NMIN(50),
     X            NATLAG(50), DNEV(3)
        WRITE(2,11)
```

```
      READ(5,12) NKULCS, NDOLG
      IF(NKULCS.GT.50)PAUSE 07
      DO 1 I=1, NKULCS
      NATLAG(I)=0
      NMAX(I)=0
      NMIN(I)=0
      ATLAG(I)=0.0
    1 READ(5,13) KLCSSZ(I), ALSLIM(I), FLSLIM(I)
      VATLAG=0.0
      NVTLG=NDOLG
      DO 2 J=1, NDOLG
      READ(5,14) DNEV,KDOLG, ALAP
      DO 3 I=1, NKULCS
      IF(KDOLG.EQ.KLCSSZ(I)) GO TO 4
    3 CONTINUE
      WRITE(2,15) J, DNEV, KDOLG, ALAP
      NVTLG=NVTLG-1
      GO TO 2
    4 IF(ALAP.LT.ALSLIM(I).OR.ALAP.GT.FLSLIM(I)) GO TO 5
      IF(ALAP.EQ.ALSLIM(I))NMIN(I)=NMIN(I)+1
      IF(ALAP.EQ.FLSLIM(I))NMAX(I) = NMAX(I)+1
      NATLAG(I) = NATLAG(I)+1
      ATLAG(I) = ATLAG(I)+ALAP
      VATLAG = VATLAG+ALAP
      WRITE(2,16) J,DNEV, KDOLG, ALAP
    2 CONTINUE
      WRITE(2,17)
      VATLAG=VATLAG/FLOAT(NVTLG)
      DO 6 I=1, NKULCS
      IF(NATLAG(I).EQ.0) GO TO 7
```

```
        ATLAG(I)=ATLAG(I)/FLOAT(NATLAG(I))
        WRITE(2,18) KLCSSZ(I), ALSLIM(I), FLSLIM(I)
      X         ATLAG(I), NATLAG(I), NMAX(I), NMIN(I)
        GO TO 6
 7      WRITE(2,19) KLCSSZ(I), ALSLIM(I), FLSLIM(I)
 6      CONTINUE
        WRITE(2,20) NVTLG, VATLAG
        PAUSE 01
11      FORMAT(1H1, //45X, 20HVALLALATI BERHELYZET,1X,
      X 9HFELMERESE,
      X //50X, 21HA VALLALATI BERLISTA:,//30X,
      X 7HSORSZAM, 3X, 15HA DOLGOZO NEVE:, 11X,
      X 11HKULCSSZAMA:, 3X, 13HALAPFIZETESE://)
12      FORMAT(2I5)
13      FORMAT(I5, 2F6.0)
14      FORMAT(3A8, I5, F6.0)
15      FORMAT(/5X, 20HHIBAS ADAT! SORSZAM:, I4, 3X,
      X 4HNEV:, 1X, 3A8, 3X,10HKULCSSZAM:, I4, 3X,
      X 5HALAP:, F6.0, 1H-)
16      FORMAT(31X, I4, 5X, 3A8, 5X, I4, 9X, F6.0, 1H-)
17      FORMAT(1H1, //45X, 20HKULCSSZAMOK SZERINTI,1X,
      X 9HMEGOSZLAS,
      X //, 22X, 10KULCSSZAM:, 3X, 11HBERHATAROK:,
      X 7X, 16HKATEGORIA ATLAG:, 8X, 5HSZAM:, 4X
      X 4HMAX:, 5X, 4HMIN:, /84X, 9H(DOLGOZO)//)
18      FORMAT(25X, I4, 4X, 2(F6.0, 1H-), 8X, F8.2,
      X 13X, 3(I4, 5X))
19      FORMAT(25X, I4, 4X, 2(F6.0, 1H-,) 9X,1H-,
      X 9(2H.-), 5HURES-, 9(2H.-))
```

```
20    FORMAT(//5X, 19HVALLALATI BERATLAG▽, I4, 1X,
     X10HDOLGOZORA:, F8.2, 1X, 3HFT.)
      END
```

A program szervezése egészen egyszerű, áttekintése nyilván nem okoz különösebb problémát, így csupán a jelölések ismertetésére és néhány megjegyzésre szorítkozunk.

Jelölések:

KLCSSZ(I) az I-edik kulcsszám; I≦50
ALSLIM(I)............ az I-edik bérkategória alsó határa
FLSLIM(I) az I-edik bérkategória felső határa
ATLAG(I) az I-edik bérkategória bérátlaga
NMAX(I) az I-edik kategóriába tartozók közül a felső bérhatárt elérők száma
NMIN(I) az I-edik kategóriába tartozók közül az alsó bérhatáron levők száma
NATLAG(I) az I-edik kategóriába tartozó összes dolgozók száma
NKULCS a vállalatnál használatos kulcsszámok száma
NDOLG.............. a vállalat dolgozóinak száma
NVTLG a felmérés szempontjából figyelembe vett dolgozók száma
DNEV a dolgozó neve; max. 24 karakter
KDOLG a dolgozó kulcsszáma
ALAP a dolgozó alapfizetése
VATLAG a vállalati bérátlag

Három rövid megjegyzés:

1. Ha valamennyi bemenő adat hibátlan, akkor NDOLG egyenlő NVTLG-vel. Ha nem, akkor NVTLG annyival kisebb, ahány hibás adatot találtunk a feldolgozás közben.

2. A program írásakor feltételeztük, hogy egy valós változóban 8 alfanumerikus karakter helyezhető el. Ez gépi reprezentációnként változhat, ezért DNEV deklarációján esetleg módosítani kell ahhoz, hogy a 24 karaktert be tudja fogadni.

3. Átlagszámításokkal kapcsolatban sohasem szabad megfeledkezni arról, hogy a kategóriák közül egyesek üresek is lehetnek. A program erre is tartalmaz védelmet.

A program egy illusztratív eredménytáblázatát a 3. sz. mellékletben mutatjuk be.

IRODALOM

[1] Gorn, S. [szerk.]: FORTRAN vs. Basic FORTRAN. Communications of the ACM, 1964. 7. köt., 10. sz. 591. old. Magyar fordításban: MTA Számítástechnikai Központja (1968).

[2] MacCracken, D. D.: Bevezetés a FORTRAN IV programozásba. SZÁMOK, 1970.

[3] MacCracken, D. D.: Numerikus alkalmazások példatár. SZÁMOK, 1970.

[4] Rabár, M. [szerk.]: FORTRAN példatár numerikus módszerek alkalmazására. SZÁMOK, 1972.

[5] IBM System/360 Operating System FORTRAN IV (G and H) Programmer's Guide. IBM kiadvány, 1973.

[6] IBM System/360 and System/370 FORTRAN IV Language. IBM kiadvány, 1974.

[7] FORTRAN programnyelv alapszintje. Magyar Szabványügyi Hivatal, 1974.

[8] FORTRAN programnyelv középső szintje. Magyar Szabványügyi Hivatal, 1974.

[9] FORTRAN programnyelv. Magyar Szabványügyi Hivatal, 1974.

[10] Bodor, T.: FORTRAN zsebkönyv. KSH, 1975.

[11] Zalotai, L.—Seres, L.—Fejes, P.: FORTRAN programok fizikai és kémiai feladatok megoldására. Műszaki Könyvkiadó, 1978.

[12] ANSI (1978), ANSI X3.9—1978: Programming Language FORTRAN.

[13] Kaufman, R. E.: FORTRAN képeskönyv. Műszaki Könyvkiadó, 1983.

[14] Bodor, T.—Gerő, P.: Bevezetés a korszerű FORTRAN programozásba. SZÁMALK, 1983.

TÁRGYMUTATÓ

A

A specifikáció 239, 367
ABS függvény 34, 310
adatállomány 358
– hivatkozási száma 358
–, közvetlen hozzáférésű 190, 358, 377
–, soros hozzáférésű 190, 358
– vége jelzés (end of file) 207, 359
adatátviteli utasítás 189, 358
– –, közvetlen hozzáférésű 377
– –, soros hozzáférésű 195, 358
AIMAG függvény 34, 311
AINT függvény 34, 310
aktuális paraméter 130, 338
– – megadási szabályai 141
– –, &címke, mint 340
ALGAMA függvény 316
ALOG függvény 35, 312
ALOG10 függvény 35, 312
AMAX0 függvény 34, 316
AMAX1 függvény 34, 316
AMIN0 függvény 34, 316
AMIN1 függvény 34, 316
AMOD függvény 34, 310
ANSI FORTRAN 300
ARCOS függvény 314
aritmetikai értékadás 49, 319
– – konverziós szabályai 50, 320
aritmetikai kifejezések 26, 309
– – típusai 29
artimetikai műveletek 24, 306

artimetikai műveletek eredményének típusai 24, 25, 307
ARSIN függvény 314
ASA FORTRAN 11, 300
— — bővítései 301
ASSIGN utasítás 52, 321
ATAN függvény 35, 314
ATAN2 függvény 35, 314
automatikus típusdeklaráció 23, 100, 326
azonosító 15, 23, 178–183, 306
—, szegmensközi 181
—, szegmensen belüli 180

B

BACKSPACE utasítás 205, 362
bájt 303
BASIC FORTRAN 11, 296
belépési pont 347
belső standard függvények 34, 36, 178, 310–311, 355
beviteli utasítás (READ) 195, 358, 361, 380
bináris formájú ábrázolás 192
BLOCK DATA szegmens 168, 355
BLOCK DATA utasítás 169, 355

C

CABS függvény 35, 316
CALL utasítás 130, 338
CCOS függvény 35, 314
CDABS függvény 316
CDCOS függvény 314
CDEXP függvény 312
CDLOG függvény 312
CDSIN függvény 314
CDSQRT függvény 312
CEXP függvény 35, 312
ciklus 67, 323
— -mag 67, 69

ciklus-specifikáció 71
– -változó 67, 69
–, kiterjesztett hatáskörű 87
ciklusok egymásba skatulyázása 77, 323
címke 13, 52
– mint aktuális paraméter 340
címkelista 53, 58
CLOG függvény 35, 312
CMPLX függvény 34, 311
COMMENT sor 13
COMMON utasítás 158, 353
– blokk 353
– blokknév 158
COMPLEX típusdeklaráló utasítás 100, 326, 328
CONJG függvény 34, 311
CONTINUE utasítás 69
COS függvény 35, 314
COSH függvény 314
COTAN függvény 314
CSIN függvény 35, 314
CSQRT függvény 35, 312

D

D specifikáció 237, 366
DABS függvény 34, 310
DARCOS függvény 314
DARSIN függvény 314
DATA utasítás 116, 332
DATAN függvény 314
DATAN2 függvény 35, 314
DBLE függvény 34, 311
DCMPLX függvény 311
DCONJG függvény 311
DCOS függvény 35, 314
DCOSH 314
DCOTAN függvény 314
DEFINE FILE utasítás 379
deklaratív utasítások 16, 96, 325

DERF függvény 316
DERFC függvény 316
DEXP függvény 35, 312
DFLOAT függvény 310
DGAMMA függvény 316
DIM függvény 34, 311
DIMENSION utasítás 101, 330
dinamikus indexhatár 153, 352
direct access adatátviteli utasítások 377
DLGAMA függvény 316
DLOG függvény 35, 312
DLOG10 függvény 35, 312
DMAX1 függvény 34, 316
DMIN1 függvény 34, 316
DMOD függvény 35, 310
DO utasítás 69, 77
DOUBLE PRECISION utasítás 100, 330
DSIGN függvény 34, 310
DSIN függvény 35, 314
DSINH függvény 314
DSQRT függvény 35, 312
DTAN függvény 35, 314
DTANH függvény 314
duplaszó 303

E

E specifikáció 230, 364
egész típusú számkonstans 20, 305
– – – változó 23, 100, 326, 328
ekvivalenciacsoport 107, 165
end of file 195, 207, 359
END utasítás 128, 338
– – paraméter READ utasításban 358
ENDFILE utasítás 205, 362
ENTRY utasítás 346
érték szerinti paraméterátvétel 144, 343
értékadó utasítás 15, 48, 319
– –, aritmetikai 49, 319

értékadó utasítás konverziós szabályai 320
— —, logikai 49, 319
— — típuseltérései 50, 320
EQ relációjel 38
EQUIVALENCE utasítás 107, 165, 330
ERF függvény 316
ERFC függvény 316
ERR paraméter READ utasításban 358
EXP függvény 35, 312
explicit típusdeklaráció 328
EXTERNAL utasítás 171, 355

F

F specifikáció 222, 364
félszó 303
feltételes utasítás 57
feltétlen vezérlésátadás 53, 321
— GO TO utasítás 321
file 193
FIND utasítás 383
FLOAT függvény 34, 310
folytatósor 13
formális paraméter 129, 141, 164, 171, 338
— —, * karakter 339, 341
— —, / jelek közé zárt 339, 343, 344
FORMAT lista 214
— — elhelyezése tömbben 266, 370
— — és az I/O lista kölcsönhatása 260
— — utasítás 213, 363
formátum specifikációk 363
FORTRAN jelkészlet 14
FUNCTION utasítás 129, 343
— — szegmens 147, 343
függvények aktivizálása 147, 338
— aktuális paraméterei 141, 344
— belépési pontjai ENTRY 346
— formális paraméterei 150, 344
— típusa 129, 149, 343

G

G specifikáció 232, 365
GAMMA függvény 316
GE relációjel 38
gépi reprezentáció 11, 300
GO TO utasítás 53, 321
— — —, feltétlen 53, 321
— — —, kijelölt 54, 321
— — —, kiszámított 58, 322
GT relációjel 38

H

H specifikáció 244, 367
hatványozás eredményének típusa 25
— szabályai 25, 308
hexadecimális konstans 305
— mennyiségek átvitele 369
HFIX függvény 310
Hollerith konstans 119, 244, 367
— mező 215
hozzárendelt változó 378

I

I specifikáció 216, 363
IABS függvény 34, 310
IDIM függvény 34, 311
IDINT függvény 34, 310
IF utasítás 55
— —, aritmetikai 55, 321
— —, logikai 57
IFIX függvény 34, 310
IMPLICIT utasítás 326
indexes változók 65, 322
indexkifejezés 65, 322
—, általánosított 322

indexkifejezés, standard 68
INT függvény 34, 310
INTEGER utasítás 100, 326, 328
I/O lista 196
— —, egyszerű 197
— —, összetett 197
ISIGN függvény 34, 310
ismétlési tényező 117, 261
— — DATA utasításban 117
— — explicit típusdeklarációban 328
— — FORMAT specifikációban 261

J

jelkészlet 14, 306

K

karakter formájú ábrázolás 191
karakterátvitel 213, 363
karakterlánc 334
kétszeres pontosság 22, 23, 100, 305, 330
kezdőértékadás 116, 328, 332
— BLOCK DATA szegmensben 168, 355
kijelölt GO TO utasítás 54, 321
kiszámított GO TO utasítás 58, 322
komplex típus aritmetikai kifejezésben 30, 319—321
konstansok 20, 305
—, hexadecimális 305
—, szöveg- 119, 334
konverzió 24, 50, 320
konverziós specifikáció 214
— szabály értékadó utasításban 319, 320
kötetlen hozzáférés 190
közös adatmezők 127, 158, 353, 355
külső standard függvények 35, 36, 178, 312—316, 355
közvetlen hozzáférés 377
— — utasításai 379, 380, 381, 383

L

L specifikáció 238, 366
LE relációjel 38
LOGICAL utasítás 100, 326, 328
logikai érték 20, 23
— értékadás 49, 319
— IF utasítás 57, 322
— kifejezés 41
— konstans 42, 49
— műveletek 39
— típus 20
— rekord 206
LT relációjel 38

M

MAX0 függvény 34, 316
MAX1 függvény 34, 316
mező 215
— -szélesség 215
MIN0 függvény 34, 316
MIN1 függvény 34, 316
MOD függvény 34, 310

N

NAMELIST lista 372
— név 372
— utasítás 372
NE relációjel 38
nemvégrehajtható utasítások 96, 325
név szerinti paraméterátvétel 144, 343
NOT művelet 40
numerikus mező 215

O

operandus 26
OR művelet 40

P

P kiegészítő specifikáció 257
paraméter 141, 343
– -átadás 141, 343
– ENTRY utasításban 348–352
– -átvétel 141, 343
paraméterek 130
–, bemenő 132
–, kimenő 132
– megfeleltetése 141
PAUSE utasítás 90, 323
precedenciaszabály 28, 309
PRINT utasítás 362
programszegmens 128
PUNCH utasítás 361

R

READ utasítás 195, 358, 361
– –, közvetlen hozzáférésű 380
REAL függvény 34, 311
– típusdeklaráló utasítás 100, 326, 328
rekord 189
reláció 38, 318
– -jelek 38
RETURN utasítás 130, 340
RETURN *i* utasítás 340, 341
REWIND utasítás 205, 362

S

SIGN függvény 34, 310
SIN függvény 35, 314
SINH függvény 314
SNGL függvény 34, 311
soros hozzáférés 190, 358
speciális jelek (karakterek) 14
SQRT függvény 35, 312
standard függvények 34–35, 178, 355
– –, belső 34, 36, 178, 310, 311, 355
– –, külső 35, 36, 178, 312–317, 355
STOP utasítás 91, 323
STOP *n* utasítás 323
SUBROUTINE utasítás 128, 338

Sz

számkonstans 20, 305
–, egész típusú 20, 305
–, hexadecimális 305
–, kétszeres pontosságú 22, 305
–, komplex típusú 22, 305
–, valós típusú 21, 305
számok 20, 305
szegmens 96, 125
– -nyitó utasítások 129, 338
– -záró utasítás 128, 338
szegmensnevek használata paraméterként 177, 355
szerkesztési specifikáció 214, 249
–, ferde törtvonal 251
–, tabulátor beállítás 368
–, vezérlő karakterek 255, 368
–, X specifikáció 249, 367
szó 303
szóhatár 331, 353
szövegkonstans 119, 334
szubrutin 130, 338
– aktuális paraméterei 131, 339

szubrutin formális paraméterei 129, 338
– hívása 131, 339
– neve aktuális paraméterként 171, 177, 340

T

T specifikáció 368
TAN függvény 314
TANH függvény 35, 314
típusdeklaráció 100, 326
–, automatikus 23, 100, 326
– BLOCK DATA szegmensben 168, 355
–, explicit 100, 328
– függvényszegmensben 149, 343
–, implicit 326
tömb 65, 101, 322, 330
– indexei 68, 322
– elhelyezkedése a tárban 104
– mérete 65, 322
tömbdeklaráció 101, 330
tömbelem címe 104

U

utasításfüggvények 121, 335

V

valós típus 20, 100
– – aritmetikai kifejezésben 24, 30, 320
valós típusú számkonstans 21
valós számkonverzió 222–238
– – előírt számú decimális jegyre 232, 365
– –, fixpontos 222, 364
– –, kétszeres pontosságú 237, 366
– –, lebegőpontos 230, 364

változók 22, 306
— típusa 23
végrehajtható utasítások 48, 319

W

WRITE utasítás 195, 361, 381
— —, közvetlen hozzáférésű 381

X

X specifikáció 249, 367

Z

Z specifikáció 369

TARTALOM

Előszó 5

Első fejezet: A programozás automatizálása és a FORTRAN programozási nyelv 9

1. Bevezetés 9
2. A FORTRAN nyelvek kialakulása 11
3. Alapfogalmak 12

Második fejezet: Aritmetikai és logikai kifejezések. Relációk 19

1. Értéktípusok 19
2. Számok 20
3. Változók, azonosítók 22
4. Aritmetikai műveletek 24
5. Aritmetikai kifejezések 26
6. Relációk 37
7. Logikai változók, műveletek, kifejezések 39
8. Gyakorlatok 45

Harmadik fejezet: Végrehajtható utasítások 48

1. Az utasítások típusai a FORTRAN nyelvben 48
2. Értékadó utasítások 48
 a) Logikai értékadó utasítás 49
 b) Aritmetikai értékadó utasítás 49
3. Vezérlő utasítások és címkék. ASSIGN utasítás 52
 a) Feltétlen vezérlésátadás. A feltétlen és a kijelölt GO TO utasítás 53
 b) Elágaztatás előjel és nulla szerint. Az aritmetikai IF utasítás 55
 c) Feltételes utasítás. A logikai IF utasítás 57
 d) Többirányú elágaztatás. A kiszámított GO TO utasítás . . 58
 e) Példák 59

4. Indexes változók, tömbök. Indexkifejezések 65
5. Ciklusutasítások és használatuk 69
a) A DO utasítás és a CONTINUE utasítás 69
b) Ciklusok egymásba skatulyázása 77
c) Kiterjesztett hatáskörű ciklus 87
6. Megállító utasítások. A PAUSE és a STOP utasítás 90
7. Gyakorlatok 91

Negyedik fejezet: Deklaratív utasítások (Nemvégrehajtható utasítások) 96

1. A FORTRAN program szerkezete. A szegmensek 96
2. A nemvégrehajtható utasítások fajtái. Az utasítások elhelyezése a szegmenseken belül 97
3. Típus- és tömbdeklarációk 100
a) Típusdeklarációk. Az INTEGER, REAL. DOUBLE PRECISION, COMPLEX és LOGICAL utasítás ... 100
b) Tömbdeklarációk. A DIMENSION utasítás 101
c) A típus- és tömbdeklarációk kölcsönhatása 102
d) A tömbök elhelyezkedése a számológép memóriájában. A tömbelem sorszáma és címe 104
4. Az EQUIVALENCE utasítás 107
5. Kezdőérték adás. A DATA utasítás 116
a) Szövegkonstansok 119
6. Utasításfüggvények 121
7. Gyakorlatok 124

Ötödik fejezet: Több szegmensből álló FORTRAN programok 125

1. Bevezetés 125
2. Szegmensnyitó és -záró utasítások. Szegmentált programok szerkezete 128
3. Szubrutinok és függvények aktivizálása. Aktuális és formális paraméterek 130
a) SUBROUTINE szegmensek. A CALL és a RETURN utasítás 130
b) FUNCTION szegmensek és alkalmazásaik 147
4. Dinamikus indexhatárok 153
5. Közös adatmezők. A COMMON utasítás 158
6. Közös adatmezőkben szereplő változók kezdőértékének beállítása. BLOCK DATA szegmensek 168
7. Szegmensnevek alkalmazása aktuális és formális paraméterként. Az EXTERNAL utasítás 171

8. Szabályok az azonosítók használatára és az aktuális értékek meghatározására 178
9. Gyakorlatok 183

Hatodik fejezet: Adatátvitel és utasításai 189

1. Perifériális egységek és külső tárolók 189
2. A READ és WRITE utasítások szintaxisa és szemantikája 195
3. Bináris formájú átvitelek. Mágnesszalagkezelő utasítások: BACKSPACE, REWIND, ENDFILE 205
4. Karakterátvitelek. A FORMAT utasítás 213
a) Egész számok konverziója. Az I specifikáció 216
b) Fixpontos, valós számkonverzió. Az F specifikáció 222
c) Lebegőpontos, valós számkonverzió. Az E specifikáció ... 230
d) Valós számkonverzió előírt számú decimális jegyre. A G specifikáció 232
e) Kétszeres pontosságú mennyiségek átvitele. A D specifikáció 237
f) Logikai mennyiségek átvitele. Az L specifikáció 238
g) Szöveg beolvasása és kiírása. Az A specifikáció 239
h) Szöveg beolvasása és kiírása. A H specifikáció 244
i) Helykitöltés. Az X specifikáció 249
j) Új rekord megnyitása. A vessző és a ferde törtvonal mint elhatároló jelek 251
k) Vezérlő karakterek 255
l) Decimális normálótényező. A P kiegészítő specifikáció ... 257
5. A FORMAT és az I/O lista kölcsönhatása. FORMAT lista elhelyezése tömbökben 260
6. Példák teljes FORTRAN programokra 270
a) Nagyméretű mátrixok szorzása 270
b) Egy görbeillesztési feladat 278
7. Gyakorlatok 293

A) Függelék. A teljes és a szűkebb FORTRAN nyelv eltérései 296

B) Függelék. Az IBM FORTRAN IV programozási nyelv 300

Első fejezet: Az ASA és az IBM FORTRAN programozási nyelv 300

1. Bevezetés 300
2. Bővítések az ASA FORTRAN-hoz képest 301

Második fejezet: Aritmetikai és logikai kifejezések. Relációk 303

1. Értéktípusok .. 303
2. Számok ... 305
3. Változók, azonosítók 306
4. Aritmetikai műveletek 306
5. Aritmetikai kifejezések 309
6. Relációk ... 318
7. Logikai változók, műveletek, kifejezések 318

Harmadik fejezet: Végrehajtható utasítások 319

1. Az utasítások típusai az IBM FORTRAN nyelvben 319
2. Értékadó utasítások 319
3. Vezérlő utasítások és címkék. ASSIGN utasítás 321
4. Indexes változók, tömbök. Indexkifejezések 322
5. Ciklusutasítások és használatuk 323
6. Megállító utasítások. A PAUSE és a STOP utasítás 323

Negyedik fejezet: Deklaratív utasítások (Nemvégrehajtható utasítások) .. 325

1. A FORTRAN program szerkezete. A szegmensek 325
2. A nemvégrehajtható utasítások fajtái. Az utasítások elhelyezése a szegmenseken belül 325
3. Típus- és tömbdeklarációk 326
4. Az EQUIVALENCE utasítás 330
5. Kezdőérték adás. A DATA utasítás 332
6. Utasításfüggvények 335

Ötödik fejezet: Több szegmensből álló FORTRAN programok 338

2. Szegmensnyitó és -záró utasítások 338
3. Szubrutinok és függvények aktivizálása. Aktuális és formális paraméterek 338
4. Dinamikus indexhatárok 352
5. Közös adatmezők. A COMMON utasítás 353
6. Közös adatmezőkben szereplő változók kezdőértékének beállítása. BLOCK DATA szegmensek 355
7. Szegmensnevek alkalmazása aktuális és formális paraméterként. Az EXTERNAL utasítás 355

Hatodik fejezet: Adatátvitel és utasításai 358

1. Periferiális egységek és külső tárolók 358
2. A READ és WRITE utasítások szintaxisa és szemantikája 358
3. Bináris formájú átvitelek. Mágnesszalagkezelő utasítások: BACKSPACE, REWIND, ENDFILE 362
4. Karakterátvitelek. A FORMAT utasítás 363
5. A FORMAT és az I/O lista kölcsönhatása. FORMAT lista elhelyezése tömbökben 370
6. Karakterátvitel a NAMELIST utasítás használatával 372
7. A közvetlen hozzáférésű adatátvitel utasításai 377

A gyakorlatok megoldásai 384

Irodalom 416

Kiadja a Műszaki Könyvkiadó
Felelős kiadó: Fischer Herbert igazgató

84/1990 Franklin Nyomda, Budapest
Felelős vezető: Mátyás Miklós igazgató

Műszaki vezető: Kőrizs Károly
Műszaki szerkesztő: Metzker Sándor
A borítót tervezte: Kováts Tibor
A könyv ábráit rajzolta: Mészáros Éva
A könyv formátuma: Fr5
Ívterjedelme: 21,9 (A/5) + 3 db melléklet
Ábrák száma: 25
Papír minősége: 80 g ofszet
Betűcsalád és -méret: New Times, gm/gm
Azonossági szám: 61 308
Ez a könyv az 1981. évi 5. kiadás *B)* Függelékkel kiegészített utánnyomása
MŰ: 3710 – i – 8587

77 Ft